Efficient Matrix-Free Implementation and Automated Verification of Hybridizable Discontinuous Galerkin Finite Element Methods

by

Corbin Foucart

B.S. Engineering Physics
Stanford University (2015)

Submitted to the Department of Mechanical Engineering
in partial fulfillment of the requirements for the degree of

Master of Science in Mechanical Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2019

Author .....
Department of Mechanical Engineering
June 2019

Certified by .....
Pierre F.J. Lermusiaux
Professor, Department of Mechanical Engineering
Thesis Supervisor

Accepted by .....
Nicolas Hadjiconstantinou
Chairman, Department Committee on Graduate Theses

Efficient Matrix-Free Implementation and Automated Verification of Hybridizable Discontinuous Galerkin Finite Element Methods

by

Corbin Foucart

Submitted to the Department of Mechanical Engineering
on June 2019, in partial fulfillment of the
requirements for the degree of
Master of Science in Mechanical Engineering

Abstract

This work focuses on developing efficient and robust implementation methods for hybridizable discontinuous Galerkin (HDG) schemes for fluid and ocean dynamics. In the first part, we compare choices in weak formulations and their numerical consequences. We address details in making the leap from the mathematical formulation to the implementation, including the different spaces and mappings, discretization of the integral operators, boundary conditions, and assembly of the linear systems. We provide a flexible mapping procedure amenable to both quadrature-free and quadrature-based discretizations, and compare the accuracy of the two on different problem geometries. We verify the quadrature-free approach, demonstrating that optimal orders of convergence can be obtained, even on non-affine and curvilinear geometries. The second part of the work investigates the scalability of HDG schemes, identifying memory and time-to-solution bottlenecks. The form of the quadrature-free integral operators is exploited to develop a novel and efficient matrix-free approach to solving the global linear system that arises from HDG discretizations. Additional manipulations to improve numerical robustness are discussed. To mitigate the complexity of the implementation, we provide an automated and computationally efficient verification procedure for the HDG methodologies discussed, using a hierarchical approach to provide diagnostic information and isolate problems. Finally, challenges related to the effective visualization of high-order, discontinuous HDG-FEM data for fluid and ocean applications are illustrated and strategies are provided to address them.

Thesis Supervisor: Pierre F.J. Lermusiaux
Title: Professor, Department of Mechanical Engineering

Acknowledgments

> The truth will set you free. But not until it is finished with you.
> — David Foster Wallace, *Infinite Jest*

There are a number of people who have been instrumental in my completing this thesis, and to whom I am grateful.

I feel an enormous sense of gratitude to Pierre for his guidance throughout my time so far at MIT — thank you for all the ideas, advice, and vision that shaped this research project. I am continually inspired by your dedication to MSEAS, your deep understanding of an improbably high number of different research areas, and your incredible work ethic. I am similarly grateful to Chris Mirabito for his near-infinite patience and many explanations when I began my work on the HDG project — not to mention his substantial contributions proofreading and revising the details of this thesis — thanks for helping me avoid "getting burned" on a regular basis. Thanks to the legendary Pat Haley for more than living up to the title of "resident wise, old guy" and for all the help with the cluster. I am very appreciative of the Office of Naval Research for support under grants N00014-15-1-2626 (DRI-FLEAT) and N00014-18-1-2781 (DRI-CALYPSO), the Defense Advanced Research Projects Agency (DARPA) for support under grant N66001-16-C-4003 (POSYDON), and the National Science Foundation for support under grants OCE-1061160 (ShelfIT) and EAR-1520825 (NSF-ALPHA), as well as to the Massachusetts Institute of Technology for making this research experience possible.

I'd also like to thank Dr. Alexander Linke of the Weierstrass Institute for his incredible mentorship during my time in Berlin — I found your love of numerical mathematics to be a source of inspiration, and my choice to pursue research in numerical solution of PDEs was largely due to my time spent working with you.

My friends and labmates have tremendously increased the dimensionality of my life here. Abhinav, thanks for the many discussions about life, your prudent financial advice, and for understanding the importance of fried chicken in any balanced diet. Jing, thanks for your encyclopedic mathematical knowledge, your crystal-clear explanations, and all of the swim workouts in the "P-pool." Arko, thanks for all the great conversations, coffee hours, and lunches from various food trucks. Thanks to Florian for the rewarding musical moments playing Kuhlau and Schubert, and thanks to JVo for all the late nights spent debugging HDG, drinking seltzer, and listening to Yuja Wang and Joshua Bell rip through the Kreuzer Sonata. Thanks to Nate, Kevin, Evan, Mike, Nick Titelbaum, Richie, and Steve for the many poker games, nighttime glow-in-the-dark ultimate frisbee matches, hikes, parties, Blue Moon calzones, and games of deception over the years — you guys are the best. To Ravi, thank you for the midnight runs, the many stories and jokes, and for being an overall exceptional flat mate. To Akis, Chinmay, Jade, Manan, Manny, Mike, Stefano, Wael, Yukino, and all of the other lab members: thanks for making MSEAS a welcoming and fun place to work.

Lastly, I'd like to thank the members of my wonderful biological family, to whom I owe the gift of life itself! Thank you, Mom and Dad; everything I've accomplished has been in no small part due to the love, support, and guidance you've given me over the years. Thanks to my sister Abbey for keeping me grounded and for all the excellent times at the Friendly Toast, and thanks to Esme for selflessly defending me from all the menacing trees, blowing leaves, and stationary shadows of Cambridge, MA.

Contents

List of Discretization Symbols

Ω The domain of interest.

$\partial\Omega$ The boundary of the domain of interest.

$\mathcal{T}_h$ The triangulation of the domain Ω into a set of non-overlapping elements K.

K A discrete element in $\mathcal{T}_h$. That is, $\mathcal{T}_h = \cup K_i$.

∂K The boundary of an element K

e The unique HDG edge element existing between K^+ and K^-. That is, $e = \partial K^+ \cap \partial K^-$.

ε The HDG edge-space. That is, $\varepsilon = \cup \partial K$.

ε° The HDG edge-space on the interior of the domain, excluding the boundaries. That is, $\varepsilon^\circ = \varepsilon \setminus \varepsilon^\partial$.

ε^∂ The HDG edge-space on the boundary of the domain. That is, $\varepsilon^\partial = \varepsilon \cap \partial\Omega$.

Γ_D The portion of $\partial\mathcal{T}_h$ with Dirichlet boundary conditions

Γ_N The portion of $\partial\mathcal{T}_h$ with Neumann boundary conditions

$\boldsymbol{n}$ The outward-pointing unit normal from a face. $\boldsymbol{n} = \hat{\mathbf{n}}(\mathbf{x}, t)$.

Introduction

The discontinuous Galerkin finite element method (DG-FEM) was first proposed as a method to solve the steady-state neutron transport equation [67], but has since developed into an entire numerical ecosystem capable of solving a wide variety of problems in computational physics. We begin with some notes on the classification of the different portions of the DG-FEM ecosystem to provide context to the work in this thesis. Discontinuous Galerkin methods were originally developed for hyperbolic conservation laws, and have enjoyed popularity for a multitude of applications based on hyperbolic systems: acoustics [4, 79] , Maxwell's equations [17, 18], and the shallow water equations [21, 25, 29, 88], to name a few. DG-FEM approaches to solving elliptic problems using *primal methods* began with the introduction of interior penalty methods [1, 70], and was extended to the class of *mixed methods* by writing the second-order spatial derivatives as a system of first-order equations [8, 11]. The extension to elliptic problems has allowed for the more recent application of DG-FEM to advection-diffusion problems, as well as to both compressible and incompressible viscous flow problems. An examination and unified analysis of the properties and differences between these methods can be found in Arnold and Brezzi [2].

Discontinuous Galerkin finite element methods can provide high-order accuracy, can represent complex geometries, and can admit both implicit and explicit semi-discrete forms. Additionally, the discontinuous polynomial spaces in which DG-FEM solutions are sought allow for the capture of steep gradients and wave behavior, resulting in more stable and flexible methods than the classical continuous Galerkin finite element (CG-FEM) approaches to advection-dominated problems [36].

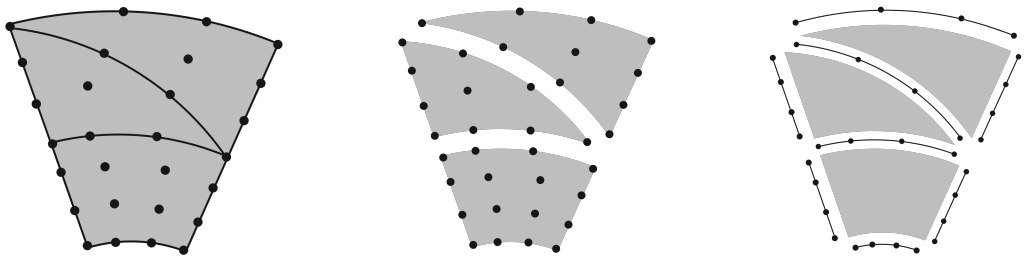


Figure 0-1: Globally coupled degrees of freedom for CG-FEM (left), DG-FEM (center), and HDG-FEM (right).

The benefits of DG-FEM, however, come at a price; namely, the increase in total degrees of freedom as a result of the discontinuous approximation spaces and element-wise decoupling of the solution. In particular, for implicit elliptic and parabolic problems, the increase in the size of the linear system can outweigh the benefits of a DG-FEM approach. Hybridizable discontinuous Galerkin (HDG) methods were first introduced for second-order elliptic

problems in Cockburn [14] in order to mitigate the cost of the discontinuous representation of the solution. In HDG schemes, the globally coupled unknowns have support on the element interfaces only, resulting in a DG approximation and convergence properties at a cost similar to classical FEM [58, 15, 40]. The differences in degree of freedom distribution for CG-FEM, DG-FEM, and HDG are schematically illustrated in Figure 0-1. In light of these benefits, recent research has extended HDG methods to a wide variety of implicit problems [59, 61, 84, 83, 85, 86, 64, 56, 60]. However, for large-scale computations, hybridization alone is often insufficient to overcome memory and time-to-solution limitations, resulting in ongoing research [73, 74, 28, 42, 27] and motivating this work.

In this thesis, we present a complete implementation for an HDG solution of a linear second-order model problem, providing both the formulation and the non-trivial leap to the implementation details. We consider a flexible and efficient treatment of boundary conditions and representation of complex geometry, as well as choices for the discretization of the integral operators that arise from an HDG formulation. We extend the implementation to a novel, efficient, matrix-free method that is both specific to the HDG methodology and that exploits the favorable properties of our particular choice of discretization. Our matrix-free approaches will address the scalability complications inherent to HDG schemes, and will provide an efficient HDG software kernel which forms the foundation for numerical schemes [61, 85, 86] used to solve fluid and ocean problems.

Chapter 1

HDG: Implementation

The purpose of this chapter is twofold. First, we provide a cursory overview of the HDG methodology applied to a simple model problem in the interest of exposition. The techniques and ideas employed in the model problem will generalize well to the more complicated problems considered in the remainder of the text. Second, although the weak form of a PDE and a choice of finite element approximation spaces is, in principle, enough to describe a scheme, in practice, each scheme includes a rich set of implementation choices; choices which can have significant computational consequences when considering performance and scalability.

Examples of these choices include: the representation of the mapping from the master element to each physical space element, which determines the types of domain geometries over which the PDE can be acceptably represented; the discretization of the integral operators (quadrature-based versus quadrature-free schemes), which can affect computational efficiency for large problems; the choice of nodal basis representation, which affects stability and conditioning for high-order polynomial solutions; and the choice of strong or weak form of the DG discretization and the handling of boundary conditions, both of which has important consequences with respect to the symmetry and positive definiteness of the discretized global linear system. Each of these choices has important implications with respect to implementation and time-to-solution. We will discuss each choice and its consequences when applied to the model problems in this chapter.

The discussion and derivation of elemental mappings and relations will be written in the context of 2D problems, because their 3D generalizations are immediately apparent and unnecessarily verbose. We will specifically note when there are significant implementation differences between 2D and 3D, and more generally, when certain implementations are appropriate for "small" or "large" problems; the latter referring to problems where required memory or computation time are at the threshold of acceptability with respect to available computational resources.

The material in section 1.1 is primarily a summary of the work contained in Nguyen [58, 57], but extended to include an explicit and flexible treatment of Dirichlet boundary conditions — compare the treatment in section 1.1 to equation (9) in Nguyen [58]. The material relating to the discretization of quadrature-free integral operators extends the ideas in Atkins [5] to the model problem, and clarifies the approach in Ueckermann and Lermusiaux (2016) [85]. The numerical experiments in section 1.3 contain novel results that extend the preliminary investigation conducted in Ueckermann (2010) [84].

The HDG methods we will consider in this context are mixed methods. Mixed meth-

ods refer to finite element methods where more than one approximation space is used to approximate the quantities of interest in the problem [11]. For example, in the case of the numerical solution of the Stokes equations with classical CG-FEM techniques, different approximation spaces can be used to treat the velocity and pressure [57]. In other mixed methods, an auxiliary variable is introduced to the differential form of the governing equations and is solved as an additional problem unknown. This approach is common in DG-like schemes with higher than first-order spatial derivatives (e.g., diffusion problems) in order to preserve problem consistency and stability [36, 90]. Such is the case in the following model problem [83, 85].

1.1 Model problem

Consider the boundary value problem (BVP) consisting of the Poisson equation, subject to the Dirichlet and Neumann boundary conditions:

$$\begin{aligned} -\nabla \cdot (\kappa \nabla u) &= f && \text{in } \Omega, \\ u &= g_D && \text{on } \Gamma_D, \\ \kappa \nabla u \cdot \boldsymbol{n} &= g_N && \text{on } \Gamma_N. \end{aligned} \tag{1.1}$$

We consider the mixed formulation of the boundary value problem by introducing an auxiliary variable $\boldsymbol{q} = \kappa \nabla u$:

$$\begin{aligned} \boldsymbol{q} - \kappa \nabla u &= 0 && \text{in } \Omega, \\ -\nabla \cdot \boldsymbol{q} &= f && \text{in } \Omega, \\ u &= g_D && \text{on } \Gamma_D, \\ \boldsymbol{q} \cdot \boldsymbol{n} &= g_N && \text{on } \Gamma_N. \end{aligned} \tag{1.2}$$

1.1.1 HDG discretization

Notation

In order to state the weak form of the problem, we will first introduce some requisite notation. We let $\mathcal{T}_h = \cup_i K_i$ be a finite collection of non-overlapping elements K_i that discretizes the entire computational domain Ω. Also, let $\partial\mathcal{T}_h = \{\partial K : K \in \mathcal{T}_h\}$ be the set of interfaces of all elements, where ∂K is the boundary of element K. For two elements sharing an edge K^+ and K^-, we define $e = \partial K^+ \cap \partial K^-$ as the edge between elements K^+ and K^-. The edges can be classified as ε° and ε^∂, the set of interior and boundary edges, respectively, with $\varepsilon = \varepsilon^\circ \cup \varepsilon^\partial$.

Figure 1-1 depicts an exploded view of a mesh containing curvilinear triangle and quadrilateral elements, as well as the interior edges ε° and boundary edges ε^∂.

The elements K^+ and K^- have outward pointing unit normals $\boldsymbol{n}^+$ and $\boldsymbol{n}^-$, respectively. The quantities $[\mathbf{a}^\pm, c^\pm]$ denote the traces of $[\mathbf{a}, c]$ on the edge e from the interior of $K^\pm$. The mean value $\{\{\bullet\}\}$ and jump $[\![\bullet]\!]$ on the interior interfaces $e \in \varepsilon^\circ$ for scalar and vector quantities are then defined as

$$\begin{aligned} \{\{\mathbf{a}\}\} &= (\mathbf{a}^+ + \mathbf{a}^-)/2, & \{\{c\}\} &= (c^+ + c^-)/2, \\ [\![\mathbf{a} \cdot \boldsymbol{n}]\!] &= \mathbf{a}^+ \cdot \boldsymbol{n}^+ + \mathbf{a}^- \cdot \boldsymbol{n}^-, & [\![c\boldsymbol{n}]\!] &= c^+ \boldsymbol{n}^+ + c^- \boldsymbol{n}^-. \end{aligned}$$

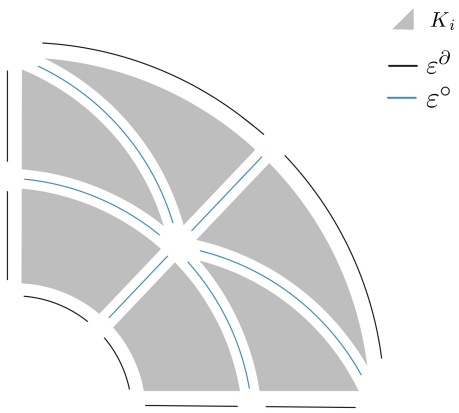


Figure 1-1

on the set of boundary interfaces $e \in \varepsilon^\partial$, with outward facing normal $\boldsymbol{n}$ on $\partial\Omega$, we define these mean and jump quantities as:

$$\{\{\mathbf{a}\}\} = \mathbf{a}, \qquad \{\{c\}\} = c,$$
$$[\![\mathbf{a} \cdot \boldsymbol{n}]\!] = \mathbf{a} \cdot \boldsymbol{n}, \qquad [\![c\boldsymbol{n}]\!] = c\boldsymbol{n}.$$

We remark that under the definitions listed above, we have the important relation between edge quantities viewed on each element K, and the mean and jump quantities viewed on the interfaces between elements. For a scalar trace $\boldsymbol{\phi}$ and vector $\boldsymbol{q}$,

$$\sum_{K\in\Omega} \int_{\partial K} (\boldsymbol{\phi}_K \boldsymbol{q}_K) \cdot \boldsymbol{n} \, d\partial K = \int_\varepsilon [\![\boldsymbol{\phi}]\!] \, \{\{\boldsymbol{q}\}\} \, d\varepsilon + \int_{\varepsilon^\circ} \{\{\boldsymbol{\phi}\}\} \, [\![\boldsymbol{q}]\!] \, d\varepsilon. \tag{1.3}$$

Lastly, we define the inner products over continuous domains $D \in \mathbb{R}^d$ and $\partial D \in \mathbb{R}^{d-1}$ as

$$(\mathbf{a}, \mathbf{b})_D = \int_D \mathbf{a} \cdot \mathbf{b} \, \mathrm{d}D, \qquad (c, d)_D = \int_D cd \, \mathrm{d}D, \tag{1.4}$$
$$\langle \mathbf{a}, \mathbf{b} \rangle_{\partial D} = \int_{\partial D} \mathbf{a} \cdot \mathbf{b} \, \mathrm{d}\partial D, \qquad \langle c, d \rangle_{\partial D} = \int_{\partial D} cd \, \mathrm{d}\partial D, \tag{1.5}$$

and we define the additional inner product on the discontinuous edge space:

$$\langle \mathbf{a}, \boldsymbol{b} \rangle_\varepsilon = \sum_{e\in\varepsilon} \langle \mathbf{a}, \boldsymbol{b} \rangle_e, \qquad \langle c, d \rangle_\varepsilon = \sum_{e\in\varepsilon} \langle c, d \rangle_e. \tag{1.6}$$

for vector or scalar functions $\mathbf{a}$, c defined on ε.

Approximation spaces

Let $\mathcal{P}^p(D)$ denote the set of polynomials of degree p on a domain D. For any element $K \in \mathcal{T}_h$, we denote $W^p(K) \equiv \mathcal{P}^p(K)$ and $\boldsymbol{V}^p(K) \equiv [\mathcal{P}^p(K)]^d$. We consider the discontinuous

finite element spaces

$$
\begin{aligned}
W_h^p &\equiv \left\{ w \in L^2(\Omega) : w\big|_K \in W^p(K) \, \forall K \in \mathcal{T}_h \right\}, \\
\boldsymbol{V}_h^p &\equiv \left\{ \boldsymbol{v} \in \left[L^2(\Omega) \right]^d : \boldsymbol{v}\big|_K \in \boldsymbol{V}^p(K) \, \forall K \in \mathcal{T}_h \right\}, \\
M_h^p &\equiv \left\{ \mu \in L^2(\varepsilon_h) : \mu\big|_e \in \mathcal{P}^p(e) \, \forall e \in \varepsilon_h \right\}.
\end{aligned}
\tag{1.7}
$$

We also consider the space $M_h^p(g_D) \equiv \{\mu \in M_h^p \colon \mu = \mathsf{P} g_D \text{ on } \Gamma_D\}$ where P denotes the L^2-projection into the space $\left\{ \mu\big|_{\partial\Omega} \, \forall \mu \in M_h^p \right\}$ [58].

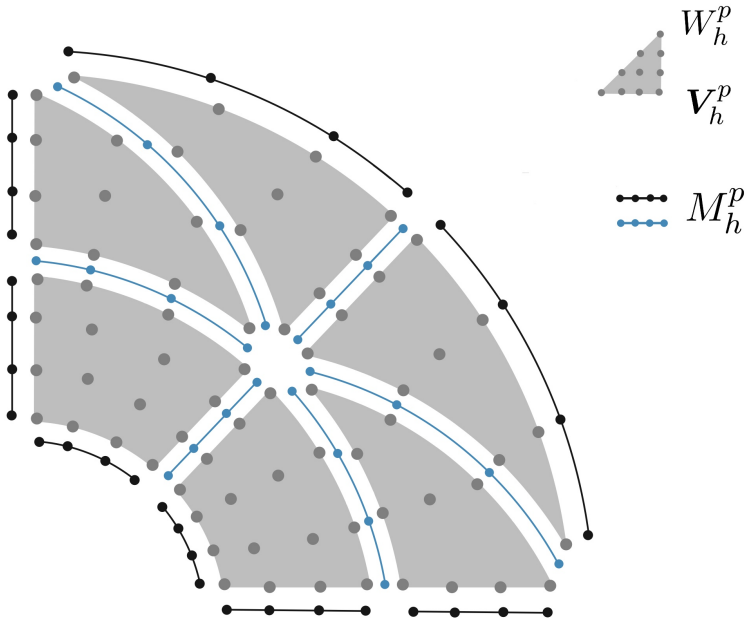


Figure 1-2

Figure 1-2 visually illustrates the spaces defined in equation (1.7) using an exploded view of the curvilinear mixed mesh depicted earlier in Figure 1-1. Boundary edges are colored differently to emphasize that the degrees of freedom on boundary edges may or may not be a problem unknown, if the boundary edge lies on Γ_N or Γ_D, respectively.

Exposition

To formulate the HDG methodology, we solve a local DG problem on each element $K \in \mathcal{T}_h$, and solve a global problem on an "element edge space", which enforces transmission conditions and domain boundary conditions. The global solution is then supplied as boundary data to each element locally as boundary conditions (problem data), from which the interior solution can be reconstructed by solving each of the element-local systems.

Formulation

We consider the model problem (1.2). On each $K \in \mathcal{T}_h$, for the given data $f\big|_K$ and $\widehat{u}_h\big|_{\partial K}$, we seek $(\boldsymbol{q}_h, u_h) \in \boldsymbol{V}^p(K) \times W^p(K)$ as the solution of the element-local problem

$$\begin{aligned} \left(\kappa^{-1}\boldsymbol{q}_h,\, \boldsymbol{v}\right)_K + (u_h,\, \nabla\cdot\boldsymbol{v})_K - \langle \widehat{u}_h,\, \boldsymbol{v}\cdot\boldsymbol{n}\rangle_{\partial K} &= 0 \qquad && \forall \boldsymbol{v} \in \boldsymbol{V}^p(K), \\ (\boldsymbol{q}_h,\, \nabla w)_K - \langle \widehat{\boldsymbol{q}}_h\cdot\boldsymbol{n},\, w\rangle_{\partial K} &= (f,\, w)_K \qquad && \forall w \in W^p(K), \end{aligned} \tag{1.8}$$

which represents a weak variational form of the original problem in equation (1.2) multiplied by test functions $\boldsymbol{v}$ and w and integrated by parts over the element K. The choice of weak variational form is not unique, and different options are discussed at length in section 1.1.2, but we proceed using equation (1.8) without loss of generality. We define

$$\widehat{\boldsymbol{q}}_h\cdot\boldsymbol{n} \equiv \boldsymbol{q}_h\cdot\boldsymbol{n} + \tau\left(\widehat{u}_h - u_h\right) \text{ on } \partial K \tag{1.9}$$

where τ is assumed to be known. Note that since $\widehat{\boldsymbol{q}}_h$ is a function of $\widehat{u}_h$, if $\widehat{u}_h$ is known on ∂K, the element-local problem is solvable. Further, if $\widehat{u}_h$ is known on the element interfaces ε, then $\widehat{u}_h$ is also known on every element boundary ∂K and *every* element-local problem can be solved independently. To determine $\widehat{u}_h$ globally, we take $\widehat{u}_h \in M_h^p$ and impose the transmission condition that the normal component of the numerical flux $\widehat{\boldsymbol{q}}_h$ be single-valued on the element edge space:

$$\langle [\![\widehat{\boldsymbol{q}}_h\cdot\boldsymbol{n}]\!],\, \mu\rangle_{\partial\mathcal{T}_h} = \langle \mathsf{P}g_N,\, \mu\rangle_{\Gamma_N}, \qquad \forall \mu \in M_h^p\left(g_D\right). \tag{1.10}$$

By summing over all mesh elements and imposing the continuity of the normal component of the numerical flux [58], the complete weak problem can be written as follows: find $(\boldsymbol{q}_h,\, u_h, \widehat{u}_h) \in (\boldsymbol{V}_h^p,\, W_h^p,\, M_h^p)$ such that

$$\begin{aligned} \left(\kappa^{-1}\boldsymbol{q}_h,\, \boldsymbol{v}\right)_{\mathcal{T}_h} + (u_h,\, \nabla\cdot\boldsymbol{v})_{\mathcal{T}_h} - \langle \widehat{u}_h,\, \boldsymbol{v}\cdot\boldsymbol{n}\rangle_{\partial\mathcal{T}_h} &= 0 && \forall \boldsymbol{v} \in \boldsymbol{V}_h^p \\ (\boldsymbol{q}_h,\, \nabla w)_{\mathcal{T}_h} - \langle \widehat{\boldsymbol{q}}_h\cdot\boldsymbol{n},\, w\rangle_{\partial\mathcal{T}_h} &= (f,\, w)_{\mathcal{T}_h} && \forall w \in W_h^p \\ \langle \widehat{\boldsymbol{q}}_h\cdot\boldsymbol{n},\, \mu\rangle_{\partial\mathcal{T}_h\setminus\Gamma_D} + \langle \widehat{u}_h - \mathsf{P}g_D,\, \mu\rangle_{\Gamma_D} &= \langle \mathsf{P}g_N,\, \mu\rangle_{\Gamma_N} && \forall \mu \in M_h^p \end{aligned} \tag{1.11}$$

In principle, this completes the formulation of the HDG methodology for linear diffusion problems. However, we have not made precise the procedure for expressing $\widehat{\boldsymbol{q}}_h$ and $\widehat{u}_h$ in terms of the element-local data. Moreover, by more closely examining the formulation, we can gain insight into the properties and features of the method.

The global problem for $\widehat{u}_h$: characterization

An interpretation of the global problem is the parametrization of the interior unknowns to the element edge space by virtue of the element-local equations, resulting in a single global problem where the only unknown is $\widehat{u}_h$. We will now characterize this procedure.

We will refer to (1.8) as the "local solver." On any element $K \in \mathcal{T}_h$, the function

$(\boldsymbol{q}^\lambda,\, u^\lambda) \in \boldsymbol{V}^p(K) \times W^p(K)$ is the solution of the local problem

$$\begin{aligned}
\left(\boldsymbol{q}^\lambda,\, \boldsymbol{v}\right)_K + \left(u^\lambda,\, \nabla\cdot\boldsymbol{v}\right)_K - \langle \lambda,\, \boldsymbol{v}\cdot\boldsymbol{n}\rangle_{\partial K} &= 0 \quad \forall \boldsymbol{v} \in \boldsymbol{V}^p(K), \\
\left(\boldsymbol{q}^\lambda,\, \nabla w\right)_K - \left\langle \widehat{\boldsymbol{q}}^\lambda\cdot\boldsymbol{n},\, w\right\rangle_{\partial K} &= 0 \quad \forall w \in W^p(K), \\
\widehat{\boldsymbol{q}}^\lambda\cdot\boldsymbol{n} &= \boldsymbol{q}^\lambda\cdot\boldsymbol{n} - \tau\left(u^\lambda - \lambda\right) \qquad \text{on } \partial K,
\end{aligned} \tag{1.12}$$

for some choice of $\lambda \in L^2(\partial K)$. Note that this is the local solver with $f = 0$ on K, and with $\widehat{u}_h = \lambda$. Secondly, the function $(\boldsymbol{q}^f,\, u^f) \in \boldsymbol{V}^p(K) \times W^p(K)$ is the solution of the element-local problem

$$\begin{aligned}
\left(\boldsymbol{q}^f,\, \boldsymbol{v}\right)_K + \left(u^f,\, \nabla\cdot\boldsymbol{v}\right)_K &= 0 \qquad\qquad \forall \boldsymbol{v} \in \boldsymbol{V}^p(K), \\
\left(\boldsymbol{q}^f,\, \nabla w\right)_K - \left\langle \widehat{\boldsymbol{q}}^f\cdot\boldsymbol{n},\, w\right\rangle_{\partial K} &= (f,\, w)_K \quad \forall w \in W^p(K), \\
\widehat{\boldsymbol{q}}^f\cdot\boldsymbol{n} &= \boldsymbol{q}^f\cdot\boldsymbol{n} - \tau u^f \qquad\qquad \text{on } \partial K,
\end{aligned} \tag{1.13}$$

for some choice of $f \in L^2(K)$. Note that this is the local solver with data f and $\widehat{u}_h = 0$. We note that the complete solution to the local problem can be expressed as $(\boldsymbol{q}_h,\, u_h) = \left(\boldsymbol{q}^\lambda,\, u^\lambda\right) + \left(\boldsymbol{q}^f,\, u^f\right)$.

Now we can state the weak formulation of the global problem. Assume that

$$\begin{aligned}
&\text{(i) } \tau\big|_e \text{ is a strictly positive constant} \qquad \forall e \in \varepsilon_h, \\
&\text{(ii) } \nabla W^p(K) \subset \boldsymbol{V}^p(K) \qquad\qquad \forall K \in \mathcal{T}_h.
\end{aligned} \tag{1.14}$$

Then the global trace $\widehat{u}_h$ is the element of $M_h^p(g_D)$ such that

$$a_h\left(\widehat{u}_h,\, \mu\right) = \ell_h\left(\mu\right) \qquad \forall \mu \in M_h^p(g_D) \tag{1.15}$$

where

$$\begin{aligned}
a_h\left(\eta,\, \mu\right) &\equiv -\left\langle \widehat{\boldsymbol{q}}^\eta\cdot\boldsymbol{n},\, \mu\right\rangle_{\partial\mathcal{T}_h}, \\
\ell_h\left(\mu\right) &\equiv \left\langle \widehat{\boldsymbol{q}}^f\cdot\boldsymbol{n},\, \mu\right\rangle_{\partial\mathcal{T}_h} - \left\langle \mathsf{P}g_N,\, \mu\right\rangle_{\Gamma_N}.
\end{aligned} \tag{1.16}$$

We make the claim that the continuous bilinear form $a_h(\cdot,\,\cdot)$ is symmetric and positive definite on $M_h^p(0) \times M_h^p(0)$; further, symmetry follows for the discretized global problem for $\widehat{u}_h$ under certain types of discrete integral operators.

Proof (symmetry of a_h):

We now prove (inspired by the remarks in Cockburn [12]) that the continuous bilinear form a_h is symmetric. By definition,

$$\begin{aligned}
a_h\left(\mu,\, \eta\right) &\equiv -\left\langle \mu,\, \widehat{\boldsymbol{q}}^\eta\cdot\boldsymbol{n}\right\rangle_{\partial\mathcal{T}_h} \\
&= -\sum_{K\in\mathcal{T}_h}\left[\left\langle \mu,\, \widehat{\boldsymbol{q}}^\eta\cdot\boldsymbol{n}\right\rangle_{\partial K}\right] \\
&= -\sum_{K\in\mathcal{T}_h}\left[\left\langle \mu,\, \boldsymbol{q}^\eta\cdot\boldsymbol{n}\right\rangle_{\partial K} - \left\langle \mu,\, \tau\left(u^\eta - \eta\right)\right\rangle_{\partial K}\right], \quad \text{by definition of } \widehat{\boldsymbol{q}}^\eta.
\end{aligned}$$

From the first equation of the local solver (1.12), we have that

$$\langle \mu,\, \boldsymbol{v}\cdot\boldsymbol{n}\rangle_{\partial K} = (\boldsymbol{q}^{\mu},\, \boldsymbol{v})_K + (u^{\mu},\, \nabla\cdot\boldsymbol{v})_K$$

for any choice of $\boldsymbol{v}\in\boldsymbol{V}^p(K)$ and $\mu\in L^2(\partial K)$. Then, in particular, the relation must hold for the choice of $\boldsymbol{v}=\boldsymbol{q}^{\eta}$, yielding

$$\langle \mu,\, \boldsymbol{q}^{\eta}\cdot\boldsymbol{n}\rangle_{\partial K} = (\boldsymbol{q}^{\mu},\, \boldsymbol{q}^{\eta})_K + (u^{\mu},\, \nabla\cdot\boldsymbol{q}^{\eta})_K\,.$$

Therefore we may write

$$a_h(\mu,\,\eta) = -\sum_{K\in\mathcal{T}_h}\left[(\boldsymbol{q}^{\mu},\,\boldsymbol{q}^{\eta})_K + (u^{\mu},\,\nabla\cdot\boldsymbol{q}^{\eta})_K - \langle\mu,\,\tau\,(u^{\eta}-\eta)\rangle_{\partial K}\right]. \tag{1.17}$$

On the other hand, integration by parts gives over each element K

$$\begin{aligned}(u^{\mu},\,\nabla\cdot\boldsymbol{q}^{\eta})_K &= \langle u^{\mu},\,\boldsymbol{q}^{\eta}\cdot\boldsymbol{n}\rangle_{\partial K} - (\nabla u^{\mu},\,\boldsymbol{q}^{\eta})_K\\ &= \langle u^{\mu},\,\tau\,(u^{\eta}-\eta)\rangle_{\partial K}\,,\end{aligned} \tag{1.18}$$

where the second equality follows from the second equation of the local solver (1.12), since we have

$$(\boldsymbol{q}^{\eta},\,\nabla w)_K - \langle\widehat{\boldsymbol{q}}^{\eta}\cdot\boldsymbol{n},\,w\rangle_{\partial K} = 0$$

for all $w\in W^p(K)$, so it must hold for the particular choice of $w=u^{\mu}$, hence

$$(\nabla u^{\mu},\,\boldsymbol{q}^{\eta})_K = \langle u^{\mu},\,\widehat{\boldsymbol{q}}^{\eta}\cdot\boldsymbol{n}\rangle_{\partial K} = \langle u^{\mu},\,\boldsymbol{q}^{\eta}\cdot\boldsymbol{n}\rangle_{\partial K} - \langle u^{\mu},\,\tau\,(u^{\eta}-\eta)\rangle_{\partial K}\,,$$

which we can substitute into (1.18), justifying the second equality. Substituting (1.18) into the expansion of $a_h(\mu,\,\eta)$ given by (1.17) yields

$$\begin{aligned}a_h(\mu,\,\eta) &= -\sum_{K\in\mathcal{T}_h}\left[(\boldsymbol{q}^{\mu},\,\boldsymbol{q}^{\eta})_K + \langle u^{\mu}-\mu,\,\tau\,(u^{\eta}-\eta)\rangle_{\partial K}\right]\\ &= -(\boldsymbol{q}^{\mu},\,\boldsymbol{q}^{\eta})_{\mathcal{T}_h} - \langle u^{\mu}-\mu,\,\tau\,(u^{\eta}-\eta)\rangle_{\partial\mathcal{T}_h}\,,\end{aligned}$$

from where it is immediately clear that $a_h(\mu,\,\eta)=a_h(\eta,\,\mu)$ and we have proved symmetry of the continuous operator. □

As an additional remark, if numerical quadrature is used, we can directly show symmetry of the discrete operator, since quadrature sums are of the form

$$\int_K f^{\mu}(\boldsymbol{x})\,f^{\eta}(\boldsymbol{x})\;d\boldsymbol{x} \approx \sum_{q=0}^{n-1} w_q f^{\mu}(\boldsymbol{x}_q)\,f^{\eta}(\boldsymbol{x}_q)\,,$$

which preserves symmetry. Importantly, if another approach is used to discretize the integral, guarantees of symmetry for the discrete bilinear form are not retained. The continuous operator must be discretized in a symmetric way.

The global problem for $\widehat{u}_h$: weak form

We now discuss the discrete equivalent of the continuous approach above whereby we eliminate the variables $\boldsymbol{q}_h$ and u_h in equation (1.11) and form a single equation for $\widehat{u}_h$, a pro-

cedure we will refer to as "static condensation." For the purposes of handling the Dirichlet boundary conditions, we introduce a new unknown $\lambda_h \in M_h^p(0)$ that vanishes on Γ_D such that

$$\widehat{u}_h = \begin{cases} \lambda_h, & \text{on } \partial\mathcal{T}_h \setminus \Gamma_D, \\ \mathsf{P} g_D, & \text{on } \Gamma_D. \end{cases} \tag{1.19}$$

Substituting the flux definitions (1.9) and (1.19) into (1.11), we can rewrite the problem as: find $(\boldsymbol{q}_h,\, u_h,\, \lambda_h) \in (\boldsymbol{V}_h^p,\, W_h^p,\, M_h^p(0))$ such that

$$\begin{aligned} \left(\kappa^{-1}\boldsymbol{q}_h,\, \boldsymbol{v}\right)_{\mathcal{T}_h} + (u_h,\, \nabla\cdot\boldsymbol{v})_{\mathcal{T}_h} - \langle\lambda_h,\, \boldsymbol{v}\cdot\boldsymbol{n}\rangle_{\partial\mathcal{T}_h} &= \langle \mathsf{P} g_D,\, \boldsymbol{v}\cdot\boldsymbol{n}\rangle_{\Gamma_D} \\ -(\nabla\cdot\boldsymbol{q}_h,\, w)_{\mathcal{T}_h} + \langle\tau u_h,\, w\rangle_{\partial\mathcal{T}_h} - \langle\tau\lambda_h,\, w\rangle_{\partial\mathcal{T}_h} &= (f,\, w)_{\mathcal{T}_h} + \langle\tau\mathsf{P} g_D,\, w\rangle_{\Gamma_D} \\ \langle\boldsymbol{q}_h\cdot\boldsymbol{n},\, \mu\rangle_{\partial\mathcal{T}_h\setminus\Gamma_D} - \langle\tau u_h,\, \mu\rangle_{\partial\mathcal{T}_h\setminus\Gamma_D} + \langle\tau\lambda_h,\, \mu\rangle_{\partial\mathcal{T}_h\setminus\Gamma_D} &= \langle\mathsf{P} g_N,\, \mu\rangle_{\Gamma_N} \end{aligned} \tag{1.20}$$

for all $(\boldsymbol{v},\, w, \mu) \in (\boldsymbol{V}_h^p,\, W_h^p,\, M_h^p(0))$. Note that $\mu \in M_h^p(0)$ removes all global basis functions on Γ_D from the unknowns of the system. The Dirichlet boundary conditions are enforced directly by the definition of $\widehat{u}_h$ in (1.19).

Following the approach in Nguyen et al. [58], we define the following linear operators and bilinear forms:

$$\begin{aligned} a(\boldsymbol{q},\, \mathbf{v}) &= \left(\kappa^{-1}\boldsymbol{q},\, \mathbf{v}\right)_{\mathcal{T}_h} & & \\ b(u, \mathbf{v}) &= (u,\, \nabla\cdot\mathbf{v})_{\mathcal{T}_h} & & \\ c(\lambda, \boldsymbol{v}) &= \langle\lambda,\, \mathbf{v}\cdot\boldsymbol{n}\rangle_{\partial\mathcal{T}_h} & r(\mathbf{v}) &= \langle\mathsf{P} g_D,\, \mathbf{v}\cdot\boldsymbol{n}\rangle_{\mathcal{T}_h} \\ d(u, \mathsf{w}) &= \langle\tau u,\, \mathsf{w}\rangle_{\partial\mathcal{T}_h} & f(\mathsf{w}) &= (f,\, \mathsf{w})_{\mathcal{T}_h} + \langle\tau\mathsf{P} g_D,\, \mathsf{w}\rangle_{\Gamma_D} \\ e(\lambda, \mathsf{w}) &= \langle\tau\lambda,\, \mathsf{w}\rangle_{\partial\mathcal{T}_h} & \ell(\mu) &= \langle\mathsf{P} g_N,\, \mu\rangle_{\Gamma_N} \\ g(u, \mu) &= \langle\tau u,\, \mu\rangle_{\partial\mathcal{T}_h} & & \\ h(\lambda, \mu) &= \langle\tau\lambda,\, \mu\rangle_{\partial\mathcal{T}_h} & & \end{aligned} \tag{1.21}$$

and we can write equation (1.20) compactly as

$$\begin{aligned} a(\boldsymbol{q}_h,\, \mathbf{v}) + b(u_h,\, \mathbf{v}) - c(\lambda_h,\, \mathbf{v}) &= r(\mathbf{v}), \\ -b(\mathsf{w},\, \boldsymbol{q}_h) + d(u_h,\, \mathsf{w}) - e(\lambda_h,\, \mathsf{w}) &= f(\mathsf{w}), \\ c(\boldsymbol{q}_h, \mu) - g(u_h,\, \mu) + h(\lambda_h,\, \mu) &= \ell(\mu), \end{aligned} \tag{1.22}$$

which represents the weak form of the problem. We could perform the discretization of (1.20) directly, which would lead to the linear system

$$\begin{bmatrix} A & B & -C \\ -B^T & D & -E \\ C^T & -G & H \end{bmatrix} \begin{bmatrix} Q \\ U \\ \Lambda \end{bmatrix} = \begin{bmatrix} R \\ F \\ L \end{bmatrix}. \tag{1.23}$$

If we were to assemble the discretized linear system (1.23) with the unknowns arranged in the order $[(q_h,\, u_h)_K\ \forall K \in \mathcal{T}_h,\, \lambda_h]$, the linear system would have features as shown in Figure 1-3, where Q, U, Λ are the vectors of degrees of freedom for $\boldsymbol{q}_h$, u_h, λ_h, respectively. The important feature is that elemental unknowns $\boldsymbol{q}_h$ and u_h correspond to a block-diagonal

structure due to the discontinuous nature of the approximation spaces [1].

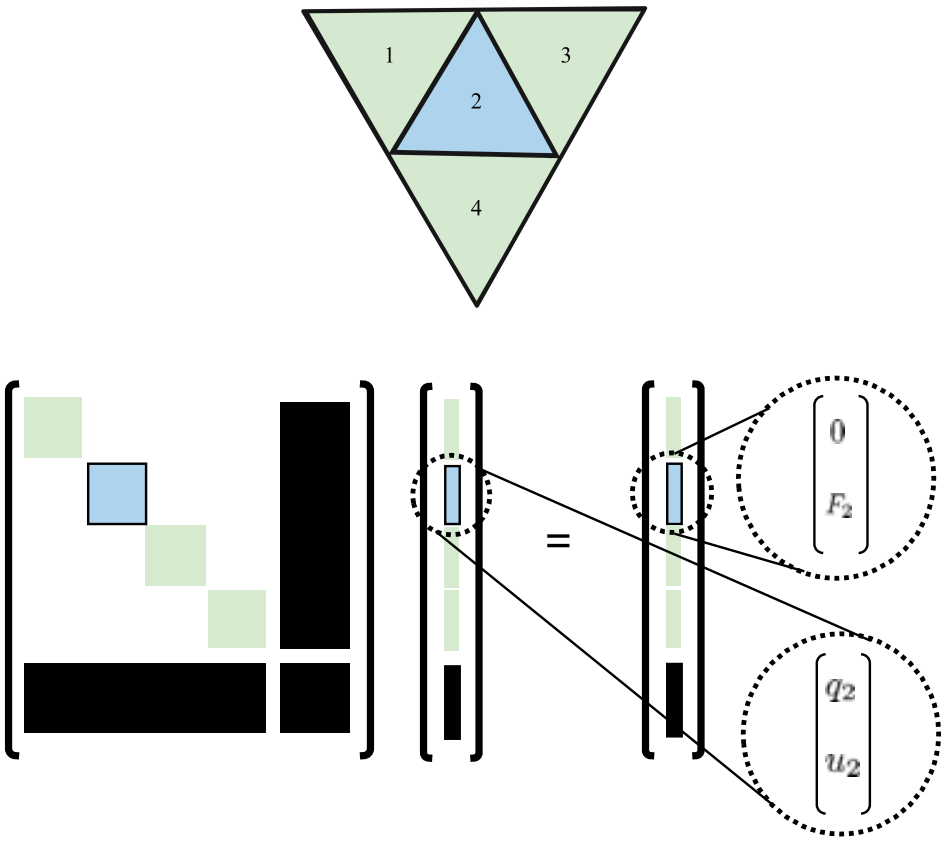


Figure 1-3: Direct discretization of (1.23) (bottom) with zero Dirichlet boundary conditions ($g_D = 0$) for a four-element mesh (top). Note that the unknown vector of the system is ordered $(\boldsymbol{q}_h, u_h)_K \; \forall K$ (green, blue), followed by λ_h (black).

In particular,

$$\begin{bmatrix} A & B \\ -B^T & D \end{bmatrix} \tag{1.24}$$

is block diagonal, which implies that Q and U can be eliminated on each element, resulting in a linear system in terms of Λ alone. To perform this elimination, we write the system (1.23) as:

[1] To see this, take the inner product $\left(\kappa^{-1}\boldsymbol{q}_h, \boldsymbol{v}\right)_{\mathcal{T}_h}$, without loss of generality. Choose a nodal basis of $\boldsymbol{v}$ such that $\boldsymbol{q}_h = \sum_i q_i\theta_i$ for global basis functions θ_i. Then the quantity $\left(\kappa^{-1}q_i\theta_i, \theta_j\right)_{\mathcal{T}_h}$ can *only* be nonzero if θ_i and θ_j are on the same element K, due to the definition of V_h^p.

$$\begin{bmatrix} A & B \\ -B^T & D \end{bmatrix} \begin{bmatrix} Q \\ U \end{bmatrix} - \begin{bmatrix} C \\ E \end{bmatrix} \Lambda = \begin{bmatrix} R \\ F \end{bmatrix},$$
$$\begin{bmatrix} C^T & -G \end{bmatrix} \begin{bmatrix} Q \\ U \end{bmatrix} + H\Lambda = L.$$

Since, by eliminating the block system for Q and U,

$$\begin{bmatrix} Q \\ U \end{bmatrix} = \begin{bmatrix} A & B \\ -B^T & D \end{bmatrix}^{-1} \left(\begin{bmatrix} R \\ F \end{bmatrix} + \begin{bmatrix} C \\ E \end{bmatrix} \Lambda \right), \tag{1.25}$$

substitution yields a single equation in terms of Λ:

$$\begin{bmatrix} C^T & -G \end{bmatrix} \begin{bmatrix} A & B \\ -B^T & D \end{bmatrix}^{-1} \left(\begin{bmatrix} R \\ F \end{bmatrix} + \begin{bmatrix} C \\ E \end{bmatrix} \Lambda \right) + H\Lambda = L,$$

forming the linear system $\mathbb{K}\Lambda = \mathbb{F}$, where

$$\begin{aligned} \mathbb{K} &= H + \begin{bmatrix} C^T & -G \end{bmatrix} \begin{bmatrix} A & B \\ -B^T & D \end{bmatrix}^{-1} \begin{bmatrix} C \\ E \end{bmatrix}, \\ \mathbb{F} &= L - \begin{bmatrix} C^T & -G \end{bmatrix} \begin{bmatrix} A & B \\ -B^T & D \end{bmatrix}^{-1} \begin{bmatrix} R \\ F \end{bmatrix}. \end{aligned} \tag{1.26}$$

As is typical in finite element implementations, these contributions are computed for each element (independently of the other elements) and are assembled to form the global linear system $\mathbb{K}\Lambda = \mathbb{F}$.

Once the linear system has been solved, Q and U can be computed directly from Λ by a modification of equation (1.25). We refer to this step as the "reconstruction" of the unknowns $\boldsymbol{q}_h$ and u_h. Note that $\widehat{u}_h$ contains both the computed unknown fluxes λ_h and the boundary conditions $\mathsf{P}g_D$[2]. Hence, we can write

$$\begin{bmatrix} Q \\ U \end{bmatrix} = \begin{bmatrix} A & B \\ -B^T & D \end{bmatrix}^{-1} \left(\begin{bmatrix} 0 \\ F \end{bmatrix} + \begin{bmatrix} C \\ E \end{bmatrix} \widehat{u}_h \right), \tag{1.27}$$

Where $F = (f, \mathsf{w})_{\mathcal{T}_h}$ in equation (1.27) rather than as defined in equation (1.21) — a slight abuse of notation.[3] We take a moment to remark upon the form of the elemental contributions $\mathbb{K}$ and right-hand side contributions $\mathbb{F}$, since their construction and application will have important implications for iterative solution techniques. The matrix inverse present in both $\mathbb{K}$ and $\mathbb{F}$ represents the parametrization of the element-local degrees of freedom to the global edge space, and is fundamental to the HDG methodology—such an

[2] Alternatively, if g_D is used in $\hat{u}$ rather than the L^2 projection $\mathsf{P}g_D$, the reconstructed solution will obey the boundary conditions exactly, but this can pollute the convergence order if g_D lies outside the polynomial space.

[3] This is because the Dirichlet information g_D is encapsulated in $\widehat{u}_h$.

inverse is not required for continuous Galerkin schemes. Assuming a static mesh in time, this inverse ought to be computed once and stored (ideally, in a memory-friendly way), and applied when forming either $\mathbb{K}$ or $\mathbb{F}$.

Boundary condition treatment

We take a moment to address treatment of Dirichlet boundary conditions in the formation of the linear system. Due to definition (1.19), all boundary data appears only on the right hand side of equation (1.20). This choice is important to the structure of the discretized linear system for λ_h.

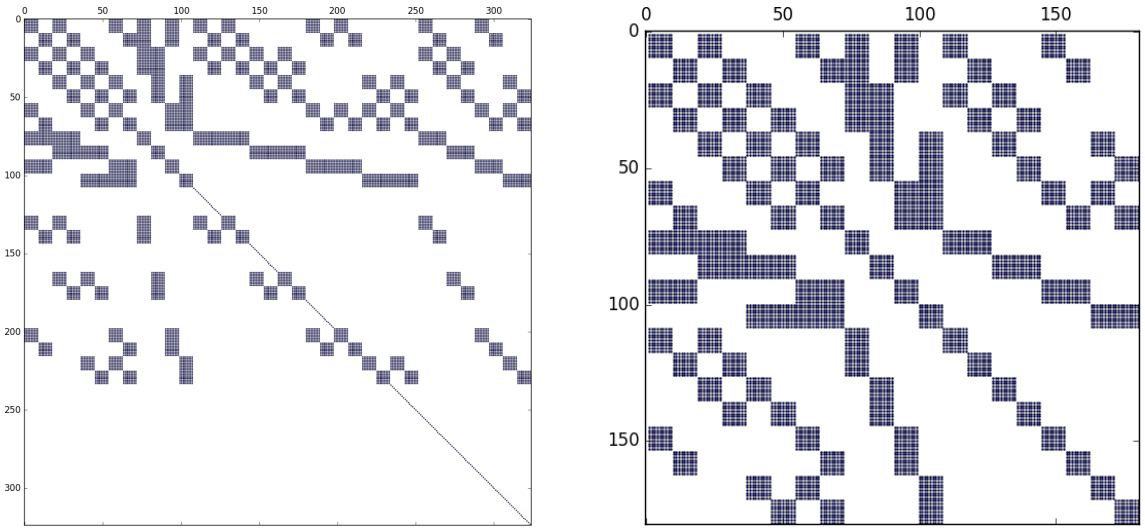


Figure 1-4: Sparsity patterns of $\mathbb{K}$ with implicit (left) and explicit (right) treatment of Dirichlet boundary conditions.

Deferring the details of how to discretize the continuous operators for the moment, if definition (1.19) is not made, and λ_h is instead sought in the space M_h^p, the Dirichlet boundary conditions will be treated implicitly, resulting in a nonsymmetric linear system $\mathbb{K}\Lambda = \mathbb{F}$, as shown in Figure 1-4. Therefore, this definition is not merely a frivolous detail, but has computational implications and results in a more smaller, symmetric linear system.

Assembly of the global linear system

Assembly of the global linear system from the elemental contributions $\mathbb{K}^K$ and $\mathbb{F}^K$ requires a mapping between the volume nodes and global unknowns in $M_h^p(g_D)$.

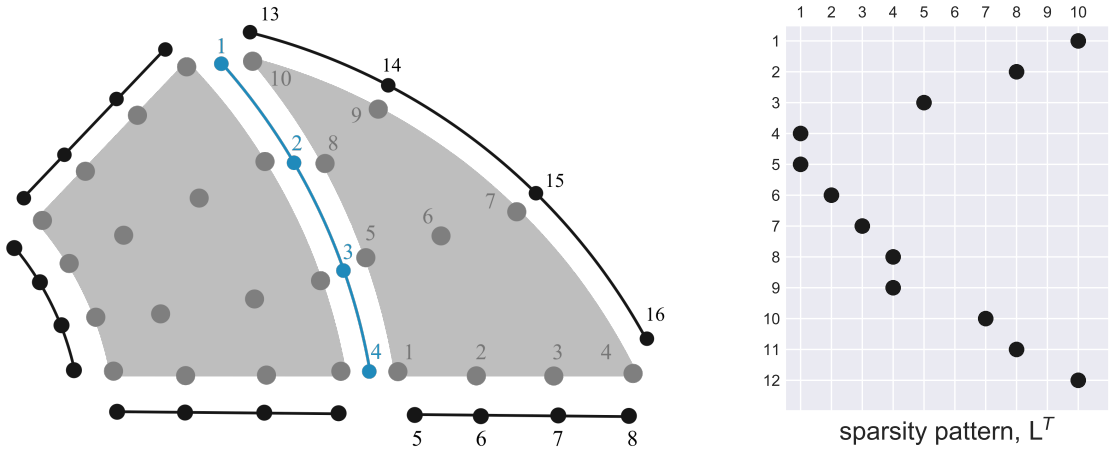
We define the "lifting" operator L, a permutation matrix that maps the nodes on the element boundary ∂K to the ordering of volume nodes. Similarly, L^T maps the volume nodes to nodes on the element boundary ∂K. The lifting operator L operator is computed once on the master element $\widehat{K}$. For details, see the implementation in Hesthaven and Warburton [36]. This operator allows for transfer of information from the elemental volume nodes to boundary nodes and vice versa. We also define the local-to-global index map $\mathcal{E}(K)$ that maps the nodes on ∂K to the edge space nodes in M_h^p for each element K.

The procedure for assembly of $\mathbb{K}$ is written in Algorithm 1, where the `meshgrid` function returns the matrices resulting from the first and second elements of the Cartesian product

Algorithm 1 Assembly of $\mathbb{K}$

$\mathbb{K} \leftarrow 0$
for $K \in \mathcal{T}_h$
    compute elemental contributions $\mathbb{K}^K$
    $\alpha \leftarrow \mathcal{E}(K) \setminus \Gamma_D$
    $J, I \leftarrow \texttt{meshgrid}(\alpha, \alpha)$
    $\mathbb{K}[I[:], J[:]] \leftarrow \mathbb{K}[I[:], J[:]] + \mathbb{K}^K[:]$
end for

$\alpha \times \alpha$ and $I[:]$ refers to the vector formed by iterating over an array in a row-wise manner. Assembly of $\mathbb{F}$ is similar.



$$\mathcal{E}(K_{tri}) = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 16 & 15 & 14 & 13 \end{bmatrix}$$

Figure 1-5: Illustration of the node numbering (left), lifting operator L^T (right), and index map $\mathcal{E}$ (bottom) for the triangular element K_{tri} in the two element mesh. Nodes on the interior edge ε° are blue, nodes on Dirichlet boundary edges $\varepsilon^\partial \in \Gamma_D$ are black, and volume nodes are grey.

To make the index map procedure explicit, we refer to the example in Figure 1-5, which depicts the node numbering around the triangular element K_{tri} for a two element mixed mesh. The boundary nodes on ∂K_{tri} are labeled from 1 to 12 counterclockwise, starting from edge node 1. The edge space nodes have a global numbering, but only the nodes on the blue interior edge are unknowns $\lambda_h \in M_h^p(g_D)$.

Implementation: serial algorithm

The steps to compute the solution $\boldsymbol{q}_h$ and u_h using the HDG methodology are summarized in Algorithm 2. We will consider various modifications to this algorithm in the interest of computational efficiency in this work. It bears repeating that computation of the element-local contributions $\mathbb{K}^K$, $\mathbb{F}^K$ as well as the elemental reconstruction of $\boldsymbol{q}_h^K$ and u_h^K can be done independently of every other element. Both steps are embarrassingly parallelizable. The solution of the global linear system $\mathbb{K}\Lambda = \mathbb{F}$ represents the portion of the algorithm where

elemental information is coupled through the edge space unknowns in the approximation space M_h^p.

Algorithm 2 HDG Algorithm

1: **for** $K \in \mathcal{T}_h$
2: compute elemental contributions $\mathbb{K}^K$, $\mathbb{F}^K$
3: **end for**
4: $\mathbb{K}$, $\mathbb{F} \leftarrow$ assemble $\mathbb{K}^K$, $\mathbb{F}^K$ for all $K \in \mathcal{T}_h$
5: $\Lambda \leftarrow$ solve $\mathbb{K}\Lambda = \mathbb{F}$
6: $\widehat{U} \leftarrow \Lambda \cup g_D$
7: **for** $K \in \mathcal{T}_h$
8: reconstruct $\boldsymbol{q}_h^K$, $u_h^K \leftarrow \begin{bmatrix} A & B \\ -B^T & D \end{bmatrix}^{-1} \left(\begin{bmatrix} 0 \\ F \end{bmatrix} + \begin{bmatrix} C \\ E \end{bmatrix} \widehat{u}_h^K \right)$.
9: **end for**

Indeed, both algorithmic features are desirable and demonstrate the advantages of the HDG approach. While a standard discontinuous Galerkin scheme involves the spatially duplicated interior degrees of freedom in the spaces W_h^p and $\boldsymbol{V}_h^p$ (depicted in Figure 1-2), the only globally-coupled unknowns in the HDG approach are the edge space degrees of freedom, drastically reducing the linear system size.

1.1.2 The "strong" and "weak" DG forms

In section 1.1.1, we noted that the choice of the weak variational form of the element local problem is not unique. Indeed, the two frequently used choices are the so-called "weak DG form" and the "strong DG form." The weak DG form is found by multiplying equation (1.2) by test functions $\boldsymbol{v}$ and w and integrating by parts over the element K, resulting in equation (1.8). The strong DG form is obtained by integrating (1.8) once more over K, yielding an equivalent form of the local problem in equation (1.28): find $(\boldsymbol{q}_h, u_h) \in \boldsymbol{V}^p(K) \times W^p(K)$ such that

$$\begin{aligned} (\boldsymbol{q}_h,\, \boldsymbol{v})_K + (\nabla u_h,\, \boldsymbol{v})_K - \langle \widehat{u}_h - u_h,\, \boldsymbol{v} \cdot \boldsymbol{n} \rangle_{\partial K} &= 0 && \forall \boldsymbol{v} \in \boldsymbol{V}^p(K), \\ (\nabla \cdot \boldsymbol{q}_h,\, w)_K - \langle (\widehat{\boldsymbol{q}}_h - \boldsymbol{q}_h) \cdot \boldsymbol{n},\, w \rangle_{\partial K} &= (f,\, w)_K && \forall w \in W^p(K). \end{aligned} \tag{1.28}$$

The two are equivalent in terms of the weak problem being posed, but differ in terms of their implementation [36]. Of course, the weak form and strong form are not the only two choices: we could integrate any term we wish by parts in the continuous representation for convenience, and indeed we have done so in the previous section. We stress that the choice of form has computational consequences in terms of both symmetry of the discretized problem, and extension to non-linear problems.

Substituting the definition of $\widehat{\boldsymbol{q}}_h$ in (1.9), and $\widehat{u}_h$ in (1.19), we find the analogous system

to (1.20). We seek $(\boldsymbol{q}_h,\, u_h,\, \lambda_h) \in (\boldsymbol{V}_h^p,\, W_h^p,\, M_h^p(0))$ such that

$$\begin{aligned}
\left(\kappa^{-1}\boldsymbol{q_h},\, \boldsymbol{v}\right)_{\mathcal{T}_h} - (\nabla u_h,\, \boldsymbol{v})_{\mathcal{T}_h} + \langle u_h,\, \boldsymbol{v}\cdot\boldsymbol{n}\rangle_{\partial\mathcal{T}_h} - \langle \lambda_h,\, \boldsymbol{v}\cdot\boldsymbol{n}\rangle_{\partial\mathcal{T}_h} &= \langle \mathsf{P}g_D,\, \boldsymbol{v}\cdot\boldsymbol{n}\rangle_{\Gamma_D}\,, \\
-(\nabla\cdot\boldsymbol{q_h},\, w)_{\mathcal{T}_h} + \langle \tau u_h,\, w\rangle_{\partial\mathcal{T}_h} - \langle \tau\lambda_h,\, w\rangle_{\partial\mathcal{T}_h} &= (f,\, w)_{\mathcal{T}_h} + \langle \mathsf{P}g_D,\, w\rangle_{\Gamma_D}\,, \\
\langle \boldsymbol{q}_h\cdot\boldsymbol{n},\, \mu\rangle_{\partial\mathcal{T}_h\setminus\Gamma_D} - \langle \tau u_h,\, \mu\rangle_{\partial\mathcal{T}_h\setminus\Gamma_D} + \langle \tau\lambda_h,\, \mu\rangle_{\partial\mathcal{T}_h\setminus\Gamma_D} &= \langle \mathsf{P}g_N,\, \mu\rangle_{\Gamma_N}\,.
\end{aligned} \tag{1.29}$$

We define a new operator $s(u,\, \mathsf{v}) = (\nabla u,\, \mathsf{v})_{\mathcal{T}_h}$ in addition to those previously defined in (1.21), and we can write the strong form system as

$$\begin{aligned}
a(\boldsymbol{q}_h,\, \mathsf{v}) - s(u_h,\, \mathsf{v}) + c(u_h - \lambda_h,\, \mathsf{v}) &= r(\mathsf{v}), \\
-b(\mathsf{w},\, \boldsymbol{q}_h) + d(u_h,\, \mathsf{w}) - e(\lambda_h,\, \mathsf{w}) &= f(\mathsf{w}), \\
c(\boldsymbol{q}_h, \mu) - g(u_h,\, \mu) + h(\lambda_h,\, \mu) &= \ell(\mu),
\end{aligned} \tag{1.30}$$

with discretization resulting in the analogy to system (1.23):

$$\begin{bmatrix} A & -S+C & -C \\ -B^T & D & -E \\ C^T & -G & H \end{bmatrix} \begin{bmatrix} Q \\ U \\ \Lambda \end{bmatrix} = \begin{bmatrix} R \\ F \\ L \end{bmatrix}. \tag{1.31}$$

Performing the same elimination, we obtain the analogous linear system $\mathbb{K}\Lambda = \mathbb{F}$, where

$$\begin{aligned}
\mathbb{K} &= H + \begin{bmatrix} C^T & -G \end{bmatrix} \begin{bmatrix} A & -S+C \\ -B^T & D \end{bmatrix}^{-1} \begin{bmatrix} C \\ E \end{bmatrix}, \\
\mathbb{F} &= L - \begin{bmatrix} C^T & -G \end{bmatrix} \begin{bmatrix} A & -S+C \\ -B^T & D \end{bmatrix}^{-1} \begin{bmatrix} R \\ F \end{bmatrix}.
\end{aligned} \tag{1.32}$$

1.1.3 Extension to time-dependent problems

Consider the time-dependent extension [57, 58] of the model problem in equation (1.2)

$$\begin{aligned}
\frac{\partial u}{\partial t} - \nabla\cdot(\kappa\nabla u) &= f \qquad \text{in } \Omega\times(0,T], \\
u &= g_D \quad \text{on } \Gamma_D\times(0,T], \\
\kappa\nabla u\cdot\boldsymbol{n} &= g_N \quad \text{on } \Gamma_N\times(0,T], \\
u &= u_0 \qquad \text{in } \Omega \text{ at } t = 0,
\end{aligned} \tag{1.33}$$

written as a system of first-order equations:

$$\begin{aligned}
\boldsymbol{q} - \kappa\nabla u &= 0 \qquad \text{in } \Omega\times(0,T], \\
\frac{\partial u}{\partial t} - \nabla\cdot\boldsymbol{q} &= f \qquad \text{in } \Omega\times(0,T], \\
u &= g_D \quad \text{on } \Gamma_D\times(0,T], \\
\boldsymbol{q}\cdot\boldsymbol{n} &= g_N \quad \text{on } \Gamma_N\times(0,T], \\
u &= u_0 \qquad \text{in } \Omega \text{ at } t = 0.
\end{aligned} \tag{1.34}$$

We modify the element-local problem with a "method of lines" approach; on each $K \in \mathcal{T}_h$, for the given data $f|_K$ and $\widehat{u}_h|_{\partial K}$, we seek $(\boldsymbol{q}_h, u_h) \in \boldsymbol{V}^p(K)\times W^p(K)$ as the solution

of the problem

$$\begin{aligned}\left(\kappa^{-1}\boldsymbol{q}_h,\, \boldsymbol{v}\right)_K + (u_h,\, \nabla\cdot\boldsymbol{v})_K - \langle \widehat{u}_h,\, \boldsymbol{v}\cdot\boldsymbol{n}\rangle_{\partial K} = 0 \qquad & \forall \boldsymbol{v}\in\boldsymbol{V}^p(K),\\ \left(\frac{\partial u_h}{\partial t},\, w\right)_K + (\boldsymbol{q}_h,\, \nabla w)_K - \langle \widehat{\boldsymbol{q}}_h\cdot\boldsymbol{n},\, w\rangle_{\partial K} = (f,\, w)_K \quad & \forall w\in W^p(K).\end{aligned} \tag{1.35}$$

The above equation can be discretized using a suitable time-marching scheme; we use a to denote a generic time-stepping coefficient, from an IMEX [82, 85], DIRK, or backward Euler scheme ($a = 1$, shown here), Making the same substitutions as above and using the operators and bilinear forms introduced in (1.21), we have the following weak problem at time-level k, without loss of generality: find $(\boldsymbol{q}_h,\, u_h,\, \lambda_h) \in (\boldsymbol{V}_h^p,\, W_h^p,\, M_h^p(0))$ such that

$$\begin{aligned} a(\boldsymbol{q}_h^k,\, \mathsf{v}) + b(u_h^k,\, \mathsf{v}) - c(\lambda_h^k,\, \mathsf{v}) \quad &= r(\mathsf{v}),\\ \frac{1}{a\Delta t^k} m(u_h^k,\, \mathsf{w}) - b(\mathsf{w},\, \boldsymbol{q}_h^k) + d(u_h^k,\, \mathsf{w}) - e(\lambda_h^k,\, \mathsf{w}) \quad &= f(\mathsf{w}) + \frac{1}{a\Delta t^k} m(u_h^{k-1},\, \mathsf{w}),\\ c(\boldsymbol{q}_h^k, \mu) - g(u_h^k,\, \mu) + h(\lambda_h^k,\, \mu) \quad &= \ell(\mu),\end{aligned} \tag{1.36}$$

for all $(\mathsf{v},\, \mathsf{w},\, \mu) \in (\boldsymbol{V}_h^p,\, W_h^p,\, M_h^p(0))$, where $m(u,\, \mathsf{w}) = (u,\, \mathsf{w})_{\mathcal{T}_h}$, and where boundary conditions and forcing are evaluated at time t^k or t^{k+1}.

1.2 Isoparametric mappings and quadrature

1.2.1 Mapping from the master element to physical space

We express the transformation from reference space to physical space as a linear combination of the basis functions of the chosen finite element space [57, 49]. The concise statement can be written:

$$\boldsymbol{x}(\boldsymbol{\xi}) = \sum_{j=1}^{n_b} \boldsymbol{x}_j^K \boldsymbol{\psi}_j(\boldsymbol{\xi}), \tag{1.37}$$

where n_b is the number of nodal shape functions defined on the master element, and $\boldsymbol{x}_j^K$ are the interpolation points on the physical space element. Note that the number of interpolation points in physical space must be the same as the number of basis functions. Therefore the 'interpolation points' are the nodal points in physical space. We can clarify the above expression by writing the explicit statement in two dimensions:

$$x(\xi,\eta) = \sum_{j=1}^{n_b} x_j^K \boldsymbol{\psi}_j(\xi,\eta), \tag{1.38}$$

$$y(\xi,\eta) = \sum_{j=1}^{n_b} y_j^K \boldsymbol{\psi}_j(\xi,\eta). \tag{1.39}$$

If we would like to map any set of points (here we choose quadrature points $\boldsymbol{\xi}_q$, but the approach is general and can be used for any set of points $\boldsymbol{\xi}_\alpha$ defined on the master element) defined on the master element to their corresponding points in physical space, we can use

the mapping directly for each quadrature point $\boldsymbol{\xi}_q$:

$$\boldsymbol{x}(\boldsymbol{\xi}_q) = \sum_{j=1}^{n_b} \boldsymbol{x}_j^K \boldsymbol{\psi}_j(\boldsymbol{\xi}_q) \qquad \rightarrow \qquad \underbrace{\begin{bmatrix} \boldsymbol{\psi}_1(\boldsymbol{\xi}_1), & \cdots, & \boldsymbol{\psi}_{n_b}(\boldsymbol{\xi}_1) \\ \vdots & & \vdots \\ \boldsymbol{\psi}_1(\boldsymbol{\xi}_q), & \cdots, & \boldsymbol{\psi}_{n_b}(\boldsymbol{\xi}_q) \end{bmatrix}}_{G} \begin{bmatrix} x_1^K, & y_1^K \\ & \\ \vdots & \vdots \\ & \\ x_{n_b}^K, & y_{n_b}^K \end{bmatrix} \tag{1.40}$$

In order to map any set of points $\boldsymbol{\xi}_\alpha$ on the master element, we need the values of the nodal shape functions evaluated at those points $\boldsymbol{\xi}_\alpha$ and the physical space nodal points $\boldsymbol{x}_j^K$. Note that in the case of mapping the nodal points on the master element to the physical space nodes, the transformation matrix G is simply the identity matrix I (nodal basis), and the physical space nodal points are recovered.

1.2.2 Derivatives of the isoparametric mapping

In this subsection, we use the explicit example of the derivatives of the isoparametric transformation in 2D, for clarity. The key idea is that since we have an explicit representation of our transformation in (1.37), we can take the derivative of the transformation with respect to master element coordinates (ξ, η):

$$\frac{\partial x}{\partial \xi} = \frac{\partial x(\xi,\eta)}{\partial \xi} = \frac{\partial}{\partial \xi}\left[\sum_{j=1}^{n_b} x_j^K \boldsymbol{\psi}_j(\xi,\eta)\right] = \sum_{j=1}^{n_b} x_j^K \frac{\partial \boldsymbol{\psi}_j}{\partial \xi}, \tag{1.41}$$

and similarly for $\frac{\partial x}{\partial \eta}$. The last expression of (1.41) is easily computable since we know the derivatives of the shape functions $\boldsymbol{\psi}_j$ on the master element with respect to the master coordinates (ξ, η). The Jacobian matrix of the transformation from the master element $\widehat{K}$ to the physical space element K at a specific point $\boldsymbol{x}_\alpha$ with corresponding point $\boldsymbol{\xi}_\alpha$ on the master element can be written

$$J\big|_{\boldsymbol{x}_\alpha} = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} \end{bmatrix}_{\boldsymbol{x}_\alpha} = \begin{bmatrix} \sum\limits_{j=1}^{n_b} x_j^K \frac{\partial \boldsymbol{\psi}_j(\boldsymbol{\xi}_\alpha)}{\partial \xi} & \sum\limits_{j=1}^{n_b} y_j^K \frac{\partial \boldsymbol{\psi}_j(\boldsymbol{\xi}_\alpha)}{\partial \xi} \\ \sum\limits_{j=1}^{n_b} x_j^K \frac{\partial \boldsymbol{\psi}_j(\boldsymbol{\xi}_\alpha)}{\partial \eta} & \sum\limits_{j=1}^{n_b} y_j^K \frac{\partial \boldsymbol{\psi}_j(\boldsymbol{\xi}_\alpha)}{\partial \eta} \end{bmatrix}, \tag{1.42}$$

which can be written compactly in matrix form as

$$J\big|_{\boldsymbol{x}_\alpha} = \begin{bmatrix} \frac{\partial \boldsymbol{\psi}_1(\boldsymbol{\xi}_\alpha)}{\partial \xi} & \cdots & \frac{\partial \boldsymbol{\psi}_{n_b}(\boldsymbol{\xi}_\alpha)}{\partial \xi} \\ \frac{\partial \boldsymbol{\psi}_1(\boldsymbol{\xi}_\alpha)}{\partial \eta} & \cdots & \frac{\partial \boldsymbol{\psi}_{n_b}(\boldsymbol{\xi}_\alpha)}{\partial \eta} \end{bmatrix} \begin{bmatrix} x_1^K, & y_1^K \\ & \\ \vdots & \vdots \\ & \\ x_{n_b}^K, & y_{n_b}^K \end{bmatrix}. \tag{1.43}$$

Note that (1.42) is directly computable from the derivatives of the shape functions, evaluated at the point of interest $\boldsymbol{\xi}_\alpha$ on the master element, as well as the physical nodal points. In practice, we evaluate the derivatives of the nodal shape functions at the quadrature points

and nodal points (for quadrature-based and quadrature-free integrals, respectively). We will denote the determinant of the matrix J as $|J|$. For every point pairing ($\boldsymbol{\xi}_\alpha$, $\boldsymbol{x}_\alpha$) on the master element and physical space, respectively, once we have computed $J(\boldsymbol{x}_\alpha)$, computing $|J|$ at $\boldsymbol{x}_\alpha$ is simply a matter of evaluating the determinant.

1.2.3 Derivatives of the isoparametric inverse mapping

We will explicitly need to compute the derivatives of the inverse mapping from physical space to reference space (discussed in 1.2.4), which arises from the need to compute terms of the form $\frac{\partial \boldsymbol{\phi}_i}{\partial x}$, i.e., the derivatives of the global nodal basis functions with respect to physical space coordinates. The difficulty is that computing and storing the derivatives of all global basis functions at every point of interest in the computational domain would be enormously expensive. Instead, we wish to map the derivatives of the global basis functions to the derivatives of the shape functions on the master element, for example, $\frac{\partial \boldsymbol{\psi}_i(\boldsymbol{\xi})}{\partial \xi}$, which are known and stored. We can link the two using the chain rule. In one spatial dimension, (1.37) can be applied with the chain rule to conclude

$$\frac{\partial \boldsymbol{\phi}_i}{\partial \xi} = \frac{\partial \boldsymbol{\phi}_i}{\partial x}\frac{\partial x}{\partial \xi} \tag{1.44}$$

since we have the functional dependence $\boldsymbol{\phi}_i(x(\xi))$. In two dimensions, we have the functional dependence shown in Figure 1-6.

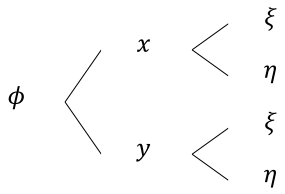


Figure 1-6: Functional dependence graph of the physical space coordinates (x, y) and the reference space coordinates (ξ, η) on the master element.

Therefore, we can write the chain rule statement [49, 57] applied over the isoparametric mapping as

$$\frac{\partial \boldsymbol{\phi}_i}{\partial \xi} = \frac{\partial \boldsymbol{\phi}_i}{\partial x}\frac{\partial x}{\partial \xi} + \frac{\partial \boldsymbol{\phi}_i}{\partial y}\frac{\partial y}{\partial \xi}, \qquad \frac{\partial \boldsymbol{\phi}_i}{\partial \eta} = \frac{\partial \boldsymbol{\phi}_i}{\partial x}\frac{\partial x}{\partial \eta} + \frac{\partial \boldsymbol{\phi}_i}{\partial y}\frac{\partial y}{\partial \eta}, \tag{1.45}$$

or, in matrix form,

$$\begin{bmatrix} \frac{\partial \boldsymbol{\phi}_i}{\partial \xi} \\ \frac{\partial \boldsymbol{\phi}_i}{\partial \eta} \end{bmatrix} = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} \end{bmatrix} \begin{bmatrix} \frac{\partial \boldsymbol{\phi}_i}{\partial x} \\ \frac{\partial \boldsymbol{\phi}_i}{\partial y} \end{bmatrix} = J \begin{bmatrix} \frac{\partial \boldsymbol{\phi}_i}{\partial x} \\ \frac{\partial \boldsymbol{\phi}_i}{\partial y} \end{bmatrix}, \tag{1.46}$$

where J is the Jacobian matrix computed at a physical space point $\boldsymbol{x}_\alpha$ in the previous section. We can directly invert this expression to find the inverse transform J^{-1}, which will allow us to compute the derivatives of the global basis functions in terms of the derivatives

of the master element shape functions:

$$\begin{bmatrix} \frac{\partial \boldsymbol{\phi}_i}{\partial x} \\ \frac{\partial \boldsymbol{\phi}_i}{\partial y} \end{bmatrix} = \begin{bmatrix} \frac{\partial \xi}{\partial x} & \frac{\partial \eta}{\partial x} \\ \frac{\partial \xi}{\partial y} & \frac{\partial \eta}{\partial y} \end{bmatrix} \begin{bmatrix} \frac{\partial \boldsymbol{\phi}_i}{\partial \xi} \\ \frac{\partial \boldsymbol{\phi}_i}{\partial \eta} \end{bmatrix} = J^{-1} \begin{bmatrix} \frac{\partial \boldsymbol{\phi}_i}{\partial \xi} \\ \frac{\partial \boldsymbol{\phi}_i}{\partial \eta} \end{bmatrix}, \tag{1.47}$$

which can be explicitly written as

$$\begin{aligned} \frac{\partial \boldsymbol{\phi}_i}{\partial x} &= \frac{\partial \boldsymbol{\phi}_i}{\partial \xi} J_{11}^{-1} + \frac{\partial \boldsymbol{\phi}_i}{\partial \eta} J_{12}^{-1}, \\ \frac{\partial \boldsymbol{\phi}_i}{\partial y} &= \frac{\partial \boldsymbol{\phi}_i}{\partial \xi} J_{21}^{-1} + \frac{\partial \boldsymbol{\phi}_i}{\partial \eta} J_{22}^{-1}. \end{aligned} \tag{1.48}$$

Extension to the general case should be apparent from the specific two-dimensional examples. Discrete computation of these $\frac{\partial \boldsymbol{\xi}}{\partial \boldsymbol{x}}$ operators will allow us to perform differentiation and integration on the master element rather than in physical space, which would be prohibitively expensive.

1.2.4 Continuous integral operators

We can write the forms of the continuous integral operators by directly substituting the inverted chain rule relations (1.48) into the physical space integral expressions [57].

Interior operators

Weak Laplacian:

$$\begin{aligned} \int_K \nabla \boldsymbol{\phi}_j \cdot \nabla \boldsymbol{\phi}_i \, d\boldsymbol{x} &= \int_K \left(\frac{\partial \boldsymbol{\phi}_i}{\partial x} \frac{\partial \boldsymbol{\phi}_j}{\partial x} + \frac{\partial \boldsymbol{\phi}_i}{\partial y} \frac{\partial \boldsymbol{\phi}_j}{\partial y} \right) d\boldsymbol{x} \\ &= \int_{\widehat{K}} \left(\frac{\partial \widehat{\boldsymbol{\phi}_i}}{\partial \xi} J_{11}^{-1} + \frac{\partial \widehat{\boldsymbol{\phi}_i}}{\partial \eta} J_{12}^{-1} \right) \left(\frac{\partial \widehat{\boldsymbol{\phi}_j}}{\partial \xi} J_{11}^{-1} + \frac{\partial \widehat{\boldsymbol{\phi}_j}}{\partial \eta} J_{12}^{-1} \right) |J| \, d\widehat{\boldsymbol{x}} \\ &\quad + \int_{\widehat{K}} \left(\frac{\partial \widehat{\boldsymbol{\phi}_i}}{\partial \xi} J_{21}^{-1} + \frac{\partial \widehat{\boldsymbol{\phi}_i}}{\partial \eta} J_{22}^{-1} \right) \left(\frac{\partial \widehat{\boldsymbol{\phi}_j}}{\partial \xi} J_{21}^{-1} + \frac{\partial \widehat{\boldsymbol{\phi}_j}}{\partial \eta} J_{22}^{-1} \right) |J| \, d\widehat{\boldsymbol{x}} \end{aligned} \tag{1.49}$$

Convection-like derivatives:

$$\begin{aligned} \int_K \nabla \boldsymbol{\phi}_i \cdot \boldsymbol{\phi}_j \, d\boldsymbol{x} &= \int_K \left(\frac{\partial \boldsymbol{\phi}_i}{\partial x} \boldsymbol{\phi}_j + \frac{\partial \boldsymbol{\phi}_i}{\partial y} \boldsymbol{\phi}_j \right) d\boldsymbol{x} \\ &= \int_{\widehat{K}} \left(\frac{\partial \widehat{\boldsymbol{\phi}_i}}{\partial \xi} J_{11}^{-1} + \frac{\partial \widehat{\boldsymbol{\phi}_i}}{\partial \eta} J_{12}^{-1} \right) \boldsymbol{\phi}_j |J| \, d\widehat{\boldsymbol{x}} + \int_{\widehat{K}} \left(\frac{\partial \widehat{\boldsymbol{\phi}_i}}{\partial \xi} J_{21}^{-1} + \frac{\partial \widehat{\boldsymbol{\phi}_i}}{\partial \eta} J_{22}^{-1} \right) \boldsymbol{\phi}_j |J| \, d\widehat{\boldsymbol{x}} \end{aligned} \tag{1.50}$$

Source term integration can be mapped as:

$$\int_K f_\Omega \boldsymbol{\phi}_i \, d\boldsymbol{x} = \int_{\widehat{K}} \widehat{f_\Omega} \, \widehat{\boldsymbol{\phi}_i} |J| \, d\widehat{\boldsymbol{x}} = \int_{\widehat{K}} f_\Omega(\boldsymbol{\xi}(\boldsymbol{x})) \widehat{\boldsymbol{\phi}_i} |J| \, d\widehat{\boldsymbol{x}}, \tag{1.51}$$

where there is some subtlety in $\widehat{f_\Omega}$, since we are integrating $f_\Omega(\boldsymbol{x})$ over some K mapped to the master element $\widehat{K}$. This inverse mapping $\boldsymbol{\xi}(\boldsymbol{x})$ maps physical space locations to their corresponding locations on the master element. While, in general, this inverse mapping is equivalent to solving a non-linear set of equations, in practice we don't need to compute the inverse map, because the points at which we evaluate the function in physical space (either nodal points or quadrature points) are mapped from the master element nodal or quadrature points. That is, the points on the master element which correspond to the physical space nodal/quadrature points are simply the nodal/quadrature points on the master element, by construction.

Edge operators

In the case of integrating over a face ε, we transform the integrand to the master edge $\widehat{\varepsilon}$. We denote basis functions over the face and master edge as θ_i and $\widehat{\theta}_i$, respectively.

$$\int_\varepsilon f \, d\varepsilon = \int_{\widehat{\varepsilon}} f |J_e| \, d\widehat{\varepsilon} \tag{1.52}$$

1.2.5 Discrete integral operators

Quadrature-based operators

We can choose a classical quadrature scheme (points and weights defined on the master element) and approximate the continuous form of the weak Laplacian operator (1.49) accordingly. We denote the master element quadrature points by $\boldsymbol{\xi}_q$ and corresponding mapped physical space quadrature points $\boldsymbol{x}_q$.

Weak Laplacian:

$$\begin{aligned}\int_K \nabla \boldsymbol{\phi}_j \cdot \nabla \boldsymbol{\phi}_i \, d\boldsymbol{x} \approx \sum_{q=1}^{n_q} &\left(\frac{\partial \widehat{\boldsymbol{\phi}_i}}{\partial \xi}(\boldsymbol{\xi}_q) J_{11}^{-1}(\boldsymbol{x}_q) + \frac{\partial \widehat{\boldsymbol{\phi}_i}}{\partial \eta}(\boldsymbol{\xi}_q) J_{12}^{-1}(\boldsymbol{x}_q) \right) \left(\frac{\partial \widehat{\boldsymbol{\phi}_j}}{\partial \xi}(\boldsymbol{\xi}_q) J_{11}^{-1}(\boldsymbol{x}_q) + \frac{\partial \widehat{\boldsymbol{\phi}_j}}{\partial \eta}(\boldsymbol{\xi}_q) J_{12}^{-1}(\boldsymbol{x}_q) \right) w_q |J| \Big|_{\boldsymbol{x}_q} \\ + \sum_{q=1}^{n_q} &\left(\frac{\partial \widehat{\boldsymbol{\phi}_i}}{\partial \xi}(\boldsymbol{\xi}_q) J_{21}^{-1}(\boldsymbol{x}_q) + \frac{\partial \widehat{\boldsymbol{\phi}_i}}{\partial \eta}(\boldsymbol{\xi}_q) J_{22}^{-1}(\boldsymbol{x}_q) \right) \left(\frac{\partial \widehat{\boldsymbol{\phi}_j}}{\partial \xi}(\boldsymbol{\xi}_q) J_{21}^{-1}(\boldsymbol{x}_q) + \frac{\partial \widehat{\boldsymbol{\phi}_j}}{\partial \eta}(\boldsymbol{\xi}_q) J_{22}^{-1}(\boldsymbol{x}_q) \right) w_q |J| \Big|_{\boldsymbol{x}_q}\end{aligned} \tag{1.53}$$

Convection-like:

$$\begin{aligned}\int_K \nabla \boldsymbol{\phi}_i \cdot \boldsymbol{\phi}_j \, d\boldsymbol{x} \approx \sum_{q=1}^{n_q} &\left(\frac{\partial \widehat{\boldsymbol{\phi}_i}}{\partial \xi}(\boldsymbol{\xi}_q) J_{11}^{-1}(\boldsymbol{x}_q) + \frac{\partial \widehat{\boldsymbol{\phi}_i}}{\partial \eta}(\boldsymbol{\xi}_q) J_{12}^{-1}(\boldsymbol{x}_q) \right) \boldsymbol{\phi}_j(\boldsymbol{\xi}_q) w_q |J| \Big|_{\boldsymbol{x}_q} \\ + \sum_{q=1}^{n_q} &\left(\frac{\partial \widehat{\boldsymbol{\phi}_i}}{\partial \xi}(\boldsymbol{\xi}_q) J_{21}^{-1}(\boldsymbol{x}_q) + \frac{\partial \widehat{\boldsymbol{\phi}_i}}{\partial \eta}(\boldsymbol{\xi}_q) J_{22}^{-1}(\boldsymbol{x}_q) \right) \boldsymbol{\phi}_j(\boldsymbol{\xi}_q) w_q |J| \Big|_{\boldsymbol{x}_q}\end{aligned} \tag{1.54}$$

Quadrature-free operators

It is possible to form integral operators without using a "traditional" quadrature scheme (points and weights chosen on the master element distinct from the nodal points) [5, 36, 85]. Note that forming the quadrature-based integral operators above involves the quantities of interest at the quadrature points on each element. However, by virtue of using nodal discontinuous Galerkin schemes, the solution coefficients are known at the nodal points; the

finite element expansion of the solution must be evaluated at the quadrature points, which can be expensive in 3D and at high order. At polynomial order p and problem dimension d, the naive evaluation complexity per hexahedral element is $\mathcal{O}\left((p+1)^{2d}\right)$ operations, which can be reduced via a sum-factorization approach to $\mathcal{O}\left(d(p+1)^{d+1}\right)$ operations [42]. However, if the integral operators can be computed using only the nodal finite element data, the cost of interpolation to the quadrature points can be obviated entirely.

Another advantage of quadrature-free schemes pertains to the evaluation of both integral operators and discrete right hand side integrals. Suppose we wish to compute the mass matrix $\mathcal{M}_K$ for integration over element K. The entries

$$\mathcal{M}_{ij} = \int_K \boldsymbol{\phi}_i(\boldsymbol{x})\boldsymbol{\phi}_j(\boldsymbol{x})d\,\boldsymbol{x} = \int_{\widehat{K}} \widehat{\boldsymbol{\phi}}_i(\boldsymbol{\xi})\widehat{\boldsymbol{\phi}}_j(\boldsymbol{\xi})|J(\boldsymbol{x})|d\,\boldsymbol{\xi} \tag{1.55}$$

computed by quadrature are approximated as

$$\sum_{q=1}^{n_q} w_q \boldsymbol{\phi}_i(\boldsymbol{\xi}_q)\boldsymbol{\phi}_j(\boldsymbol{\xi}_q)|J(\boldsymbol{x}_q)| = \widehat{\boldsymbol{\theta}_q}\,\mathrm{diag}\left(\boldsymbol{w}_q \circ |J(\boldsymbol{x}_q)|\right)\widehat{\boldsymbol{\theta}_q}^T \tag{1.56}$$

where $\circ$ denotes the Hadamard product and where $\widehat{\boldsymbol{\theta}_q}$ denotes the $n_b \times n_q$ matrix of the nodal shape functions on the master element $\widehat{K}$ evaluated at each quadrature point. While it often makes sense to compute and store the determinant vectors $|J|$ on each element, it would be inefficient to store the elemental mass matrices $\mathcal{M}_K$ for every element. Therefore, this operator must be computed (and sometimes inverted) on each element.

Consider, as an alternative, making the approximation

$$\mathcal{M}_{ij} = \int_K \boldsymbol{\phi}_i(\boldsymbol{x})\boldsymbol{\phi}_j(\boldsymbol{x})d\,\boldsymbol{x} \approx |J(\boldsymbol{x})| \int_{\widehat{K}} \widehat{\boldsymbol{\phi}}_i(\boldsymbol{\xi})\widehat{\boldsymbol{\phi}}_j(\boldsymbol{\xi})d\,\boldsymbol{\xi}, \tag{1.57}$$

an approximation which is exact in the case of a constant $|J|$ over element K.[4] Otherwise an error is incurred; however, we will show that for straight-sided elements, the error is acceptable and often dominated by discretization error. Moreover, we will show that we obtain convergence (and often to optimal order) with quadrature-free schemes and an isoparametric mapping even in the case of curved boundaries, although this represents a variational crime and the standard Galerkin error analyses are no longer valid. However, it is often the case that the curved elements are along the boundary of the domain, whereas the interior elements are straight-sided. In this case, it is always possible to treat curved elements separately and with quadrature, if necessary, while employing the quadrature-free approach on the interior straight-sided elements.

The advantage of making this approximation is that the mass matrices, M or M^{-1}, can be computed once on the master element. In the discussion above, we were careful to describe the isoparametric mapping at arbitrary spatial locations. Instead of computing the $|J|$ and J^{-1} at the quadrature points in physical space, we compute the data at the nodal points only. Then we can compute the elemental integral

$$\int_K f(\boldsymbol{x})d\,\boldsymbol{x} \approx M\left[|J(\boldsymbol{x}_n)| \circ f(\boldsymbol{x}_n)\right] \tag{1.58}$$

[4]This is the case when the mapping from the master element to physical element is linear. In 2D, mapping from the master triangle to physical triangle is always a linear transformation; a linear mapping transforms the master quadrilateral to an arbitrary parallelogram. In 3D, a linear map transforms the master hexahedron to an parallelepiped, and the master wedge to a triangular prism with parallel triangular faces.

as a matrix vector product, where $\boldsymbol{x}_n$ are the nodal points on the element, without any interpolation to quadrature points. On the other hand, the left hand side integral operator

$$\int_K (\cdot) d\,\boldsymbol{x} \approx \text{diag}(|J|) M (\cdot) \tag{1.59}$$

involves only broadcasting $|J|$ to the rows of the pre-computed master element mass matrix M. We will extensively discuss the many computational benefits of quadrature-free integral operators in Chapter 2.

Because the scheme in Ueckermann et al. [85] employs an HDG formulation using the "strong form" DG-FEM, the weak Laplacian as written above does not appear (this would not be the case in a continuous Galerkin finite element formulation). Instead, the integral derivative operators are similar to the convection-like operators in (1.50).

Convection-like:

$$\begin{aligned}\int_K \nabla \boldsymbol{\phi}_i \cdot \boldsymbol{\phi}_j \, d\boldsymbol{x} \approx & \sum_{q=1}^{n_b} \left(\frac{\partial \widehat{\boldsymbol{\phi}_i}}{\partial \xi}(\boldsymbol{\xi}_q^n) J_{11}^{-1}(\boldsymbol{x}_q^n) + \frac{\partial \widehat{\boldsymbol{\phi}_i}}{\partial \eta}(\boldsymbol{\xi}_q^n) J_{12}^{-1}(\boldsymbol{x}_q^n) \right) \boldsymbol{\phi}_j(\boldsymbol{\xi}_q^n) |J| \Big|_{\boldsymbol{x}_q^n} \\ & + \sum_{q=1}^{n_b} \left(\frac{\partial \widehat{\boldsymbol{\phi}_i}}{\partial \xi}(\boldsymbol{\xi}_q^n) J_{21}^{-1}(\boldsymbol{x}_q^n) + \frac{\partial \widehat{\boldsymbol{\phi}_i}}{\partial \eta}(\boldsymbol{\xi}_q^n) J_{22}^{-1}(\boldsymbol{x}_q^n) \right) \boldsymbol{\phi}_j(\boldsymbol{\xi}_q^n) |J| \Big|_{\boldsymbol{x}_q^n}\end{aligned} \tag{1.60}$$

We emphasize that there are no weights and all quantities of interest are evaluated at nodal points $\boldsymbol{\xi}_q^n$ or $\boldsymbol{x}_q^n$ rather than quadrature points. By analogy, we create the quadrature-free weak Laplacian operator.

Weak Laplacian (analogy):

$$\begin{aligned}\int_K \nabla \boldsymbol{\phi}_j \cdot \nabla \boldsymbol{\phi}_i \, d\boldsymbol{x} \approx & \sum_{q=1}^{n_b} \left(\frac{\partial \widehat{\boldsymbol{\phi}_i}}{\partial \xi}(\boldsymbol{\xi}_q^n) J_{11}^{-1}(\boldsymbol{x}_q^n) + \frac{\partial \widehat{\boldsymbol{\phi}_i}}{\partial \eta}(\boldsymbol{\xi}_q^n) J_{12}^{-1}(\boldsymbol{x}_q^n) \right) \left(\frac{\partial \widehat{\boldsymbol{\phi}_j}}{\partial \xi}(\boldsymbol{\xi}_q^n) J_{11}^{-1}(\boldsymbol{x}_q^n) + \frac{\partial \widehat{\boldsymbol{\phi}_j}}{\partial \eta}(\boldsymbol{\xi}_q^n) J_{12}^{-1}(\boldsymbol{x}_q^n) \right) |J| \Big|_{\boldsymbol{x}_q^n} \\ & + \sum_{q=1}^{n_b} \left(\frac{\partial \widehat{\boldsymbol{\phi}_i}}{\partial \xi}(\boldsymbol{\xi}_q^n) J_{21}^{-1}(\boldsymbol{x}_q^n) + \frac{\partial \widehat{\boldsymbol{\phi}_i}}{\partial \eta}(\boldsymbol{\xi}_q^n) J_{22}^{-1}(\boldsymbol{x}_q^n) \right) \left(\frac{\partial \widehat{\boldsymbol{\phi}_j}}{\partial \xi}(\boldsymbol{\xi}_q^n) J_{21}^{-1}(\boldsymbol{x}_q^n) + \frac{\partial \widehat{\boldsymbol{\phi}_j}}{\partial \eta}(\boldsymbol{\xi}_q^n) J_{22}^{-1}(\boldsymbol{x}_q^n) \right) |J| \Big|_{\boldsymbol{x}_q^n}\end{aligned} \tag{1.61}$$

1.2.6 Quadrature-free discretization of integral operators

We provide notes on the implementation choices for the quadrature-free operators described in (1.21) over an element $K \in \mathcal{T}_h$, following from the continuous integral forms described in section 1.2.4. Define the discrete operators

$$M_{ij} = (\boldsymbol{\phi}_i, \boldsymbol{\phi}_j)_{\widehat{K}} \quad S_{ij}^k = \left(\boldsymbol{\phi}_i, \frac{\partial \boldsymbol{\phi}_j}{\partial x_k} \right)_{\widehat{K}} \tag{1.62}$$

over the master element. We make a remark as to the vector-valued test functions. By our choice of polynomial space

$$\boldsymbol{V}_h^p \equiv \left\{ \boldsymbol{v} \in \left[L^2(\Omega) \right]^d : \boldsymbol{v}|_K \in \boldsymbol{V}^p(K) \, \forall K \in \mathcal{T}_h \right\}$$

We can view the space $\boldsymbol{V}_h^p$ as d copies of the scalar space W_h^p, which suggests how to implement the space discretely. Suppose $d = 2$. We view the unknown $\boldsymbol{q}_h = (q_x, q_y)^T$. We

can form our system of linear equations by choosing the special test functions $\mathbf{v} = (v_1, 0)^T$ then $\mathbf{v} = (0, v_2)^T$, which certainly spans the test space.

With this convenient choice of test function, inner products $(\boldsymbol{\phi}_i^k, \boldsymbol{\phi}_j^\ell)$ are only non-zero when $k = \ell$. This implies that the discrete operator A will be block diagonal—and we can assemble each vector component separately: for example,

$$\begin{aligned} a(\boldsymbol{q}, \mathbf{v}) &= \left(\kappa^{-1}\boldsymbol{q}, \mathbf{v}\right)_K \\ &= \int_{\widehat{K}} \left[\kappa^{-1}\right]^k q_i^k \widehat{\boldsymbol{\phi}}_i \widehat{\boldsymbol{\phi}}_j |J| \, d\widehat{\boldsymbol{x}} \end{aligned} \tag{1.63}$$

and so

$$[A]^k \approx \operatorname{diag}\left(\left[\kappa^{-1}\right]^k \circ |J|_K\right) M_K, \tag{1.64}$$

where $\circ$ denotes the element-wise or Hadamard product, where $[A]^k$ operates on the unknown vector q^k, and where we have made use of the fact that $M^T = M$. Note that our operator allows for an anisotropic κ which depends both on space and on coordinate direction k. The quantity $|J|$ denotes the Jacobian determinant at the nodal degrees of freedom (as opposed to at the quadrature points for a quadrature-based scheme).

$$A = \begin{bmatrix} A_x & 0 \\ 0 & A_y \end{bmatrix} \tag{1.65}$$

We take a moment to remark that one of the drawbacks using this type of quadrature-free integral operator is that the operators are not symmetric, even if they would be with quadrature. To see this, note for example that if $|J|$ (or κ) varies spatially over an element K, each block A_x, A_y in equation (1.65) will have its rows scaled by the entries of $|J|$, whereas a sum over quadrature points removes this asymmetry. This is demonstrated in Figure 1-7, which shows $A - A^T$ computed on a 2D quadrilateral element with a spatially varying $|J|$, both in the quadrature-free manner above and in a quadrature-based manner, the latter of which preserves symmetry. The result is that the elemental contributions to and, hence, the global linear system $\mathbb{K}$ will not in general be symmetric (despite the symmetry of the continuous operator as proved earlier in the chapter). We will discuss the implications of this choice in Chapter 2.

We can write the continuous operator $b(u, \mathbf{v})$ as:

$$\begin{aligned} b(u, \mathbf{v}) = (u, \nabla \cdot \mathbf{v})_K &= \left(u, \frac{\partial v_1}{\partial x} + \frac{\partial v_2}{\partial y}\right)_K \\ &= \int_{\widehat{K}} u_i \left[\left(\frac{\partial \widehat{\boldsymbol{\phi}}_j^1}{\partial \xi} J_{11}^{-1} + \frac{\partial \widehat{\boldsymbol{\phi}}_j^1}{\partial \eta} J_{12}^{-1}\right) \widehat{\boldsymbol{\phi}}_i + \left(\frac{\partial \widehat{\boldsymbol{\phi}}_j^2}{\partial \xi} J_{21}^{-1} + \frac{\partial \widehat{\boldsymbol{\phi}}_j^2}{\partial \eta} J_{22}^{-1}\right) \widehat{\boldsymbol{\phi}}_i\right] |J| \, d\widehat{\boldsymbol{x}}, \end{aligned} \tag{1.66}$$

therefore the discrete operators are formed as

$$[B_{ij}]^k = \operatorname{diag}(|J|_K) \sum_{\ell=1}^{d} S_{ij}^\ell J_{k\ell}^{-1} \qquad \Rightarrow B^k = \operatorname{diag}\left(|J|_K \circ J_{k\ell}^{-1}\right) \sum_{\ell=1}^{d} \left[S^\ell\right]^T, \tag{1.67}$$

where we note that the transpose of the stiffness matrices S^T must be used since, for

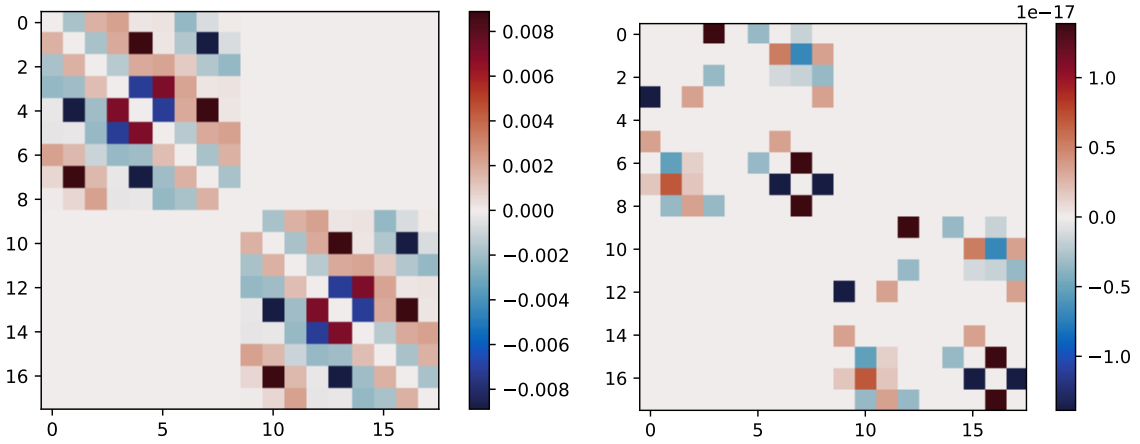


Figure 1-7: The entries of $A - A^T$ for an example of A in (1.65) assembled using the quadrature-free (left) approach and quadrature-based (right) approach for a $p = 2$ element with spatially varying $|J|$. The quadrature-based operator is symmetric to machine precision.

example,

$$\left(\widehat{\boldsymbol{\phi}}_i,\ \frac{\partial \widehat{\boldsymbol{\phi}}_j}{\partial \xi}\right) u_i J_{11}^{-1} = \operatorname{diag}\left(J_{11}^{-1}\right)\left[S^{\xi}\right]^T \boldsymbol{u}. \tag{1.68}$$

Further, note that in the case of the strong DG form, or where the operators are of the form $(u, \nabla w)_K$ or $(\nabla \cdot \boldsymbol{q}, w)_K$, matrix multiplication rules dictate that $S_{ij}u_j = S\boldsymbol{u}$ is the form of the discrete operator; incidentally, this is why (1.24) contains the B and B^T operators.

For the edge terms, we integrate over the entire boundary of the element ∂K, transforming to the boundary of the master element. The operator

$$\begin{aligned} c(\lambda, \boldsymbol{v}) = \langle \lambda,\ \mathbf{v} \cdot \boldsymbol{n} \rangle_{\partial K} &= \int_{\partial K} \lambda \left(v_1 n_1 + v_2 n_2\right)\, d\partial K, \\ &= \int_{\partial \widehat{K}} \lambda_i \widehat{\theta}_i \widehat{\theta}_j^k n_k |J_e|_K \, d\partial K \end{aligned} \tag{1.69}$$

yielding

$$C^k = \mathsf{L} \operatorname{diag}\left(|J_e|_K \circ n_k\right) M_e. \tag{1.70}$$

Similarly, the edge operator

$$d(u, \mathsf{w}) = \langle \tau u,\ \mathsf{w} \rangle_{\partial K} = \int_{\partial \widehat{K}} \tau u_i \widehat{\theta}_i \widehat{\theta}_j |J_e|_{\partial K} \, d\partial K \tag{1.71}$$

has the discrete form

$$D = \mathsf{L} \operatorname{diag}\left(\tau \circ |J_e|_K\right) M_e \mathsf{L}^T. \tag{1.72}$$

The operator $e(\lambda, \mathsf{w})$ is computed similarly. The test functions for the operators $g(u, \mu)$ and $h(\lambda, \mu)$ are the global basis functions of the edge space M_h^p (which correspond to the unknowns Λ in the global linear system). However, we compute the operators on each element, which involves performing the integrations over ∂K—the index mapping which connects $\hat{u}$ to u determines the relationship between the element edge space degrees of

freedom on ∂K and the edge space unknown vector Λ. We compute operators $g(u,\,\mu)$ and $h(\lambda,\,\mu)$ in a similar way:

$$g(u,\mu) = \langle \tau u,\, \mu \rangle_{\partial K} \tag{1.73}$$

hence

$$G = \operatorname{diag}\left(\tau \circ |J_e|_K\right) M_e \mathsf{L}^T. \tag{1.74}$$

As for the right hand side vectors, let the elemental boundary condition vectors g_D and g_N denote the element-wise products $\mathsf{P}g_D \circ \mathbf{1}_{\Gamma_D}$ and $\mathsf{P}g_N \circ \mathbf{1}_{\Gamma_N}$ on the element boundary ∂K, respectively, where $\mathbf{1}_{\Gamma_D}$ and $\mathbf{1}_{\Gamma_N}$ are the indicator functions for Γ_D and Γ_N on ∂K, respectively. Then

$$f(\mathsf{w}) = (f,\, \mathsf{w})_K + \langle \tau \mathsf{P}g_D,\, \mathsf{w} \rangle_{\Gamma_D}, \qquad \boldsymbol{f} = M\left(\boldsymbol{f}_K \circ |J|_K\right) + \mathsf{L}M_e\left(|J_e|_K \circ \tau \circ g_D\right), \tag{1.75}$$

$$r(\mathsf{v}) = \langle \mathsf{P}g_D,\, \mathbf{v}\cdot\boldsymbol{n} \rangle_{\partial K \cap \Gamma_D}, \qquad \boldsymbol{r}^k = \mathsf{L}M_e\left(|J_e|_K \circ g_D \circ n_k\right), \tag{1.76}$$

$$\ell(\mu) = \langle \mathsf{P}g_N,\, \mu \rangle_{\Gamma_N}, \qquad \boldsymbol{\ell} = M_e\left(|J_e|_K \circ g_N\right). \tag{1.77}$$

1.2.7 Discrete differentiation operators

It is often the case that we wish to take derivatives of fields explicitly, in the case of source terms and the like. In this case, we can take advantage of the fact that we have computed the $\frac{\partial \boldsymbol{\xi}}{\partial \boldsymbol{x}}$ and J^{-1} operators, which is discussed in Hesthaven and Warburton [36] in chapter 6. Consider a nodal representation $u = u_i \boldsymbol{\phi}_i$:

$$\frac{\partial u}{\partial x_k} = \frac{\partial}{\partial x_k}\left(u_i \boldsymbol{\phi}_i(\boldsymbol{x})\right) = u_i \frac{\partial \boldsymbol{\phi}_i(\boldsymbol{x})}{\partial x_k}. \tag{1.78}$$

That is, the representation of a discrete u in terms of nodal basis functions implies that the derivatives $\frac{\partial u}{\partial x_k}$ are known if the derivatives of the nodal basis functions are known. Define the differentiation matrix $\mathcal{D}^k_{ij}$ as the operator which maps the function u defined by its nodal values $u(\boldsymbol{x}_i)$ to its discrete derivatives at the same points in the k coordinate direction, $\frac{\partial u(\boldsymbol{x})}{\partial x_k} = \mathcal{D}^k u(\boldsymbol{x})$. Consider the product $\mathcal{M}\mathcal{D}^k$ where $\mathcal{M}$ is the transformed mass matrix of element K. We have:

$$\left[\mathcal{M}\mathcal{D}^k\right]_{ij} = \sum_{\ell=1}^{nb} \mathcal{M}_{i\ell}\mathcal{D}^k_{\ell j} = \int_K \boldsymbol{\phi}_i \sum_{\ell=1}^{nb} \boldsymbol{\phi}_\ell(\boldsymbol{x}_\ell) \frac{\partial \boldsymbol{\phi}_j}{\partial x_k}\bigg|_{\boldsymbol{x}_\ell} dK = \int_K \boldsymbol{\phi}_i \frac{\partial \boldsymbol{\phi}_j}{\partial x_k}\, dK = S^k_{ij}. \tag{1.79}$$

Because the nodal basis ensures $\boldsymbol{\phi}_\ell(\boldsymbol{x}_\ell) = 1$ for all ℓ, we obtain the useful relation whereby the discrete derivative operators can be computed as $\mathcal{D}_k = \mathcal{M}^{-1} S_k$. A more intuitive notion is that the matrix S contains the derivatives of the basis functions integrated over element K, and $\mathcal{M}^{-1}$ undoes the integration, leaving the pointwise derivatives of the basis functions, which can be used to take discrete derivatives using (1.78). Note that the polynomial order of the discrete derivative will be one less than that of the nodal representation, as shown in Figure 1-8. The convergence to the true derivative is generally exponential for sufficiently smooth functions; see Hesthaven and Warburton [36] for further discussion.

More generally, since we suppose that the transformation $\boldsymbol{x}(\boldsymbol{\xi})$ is invertible, with inverse

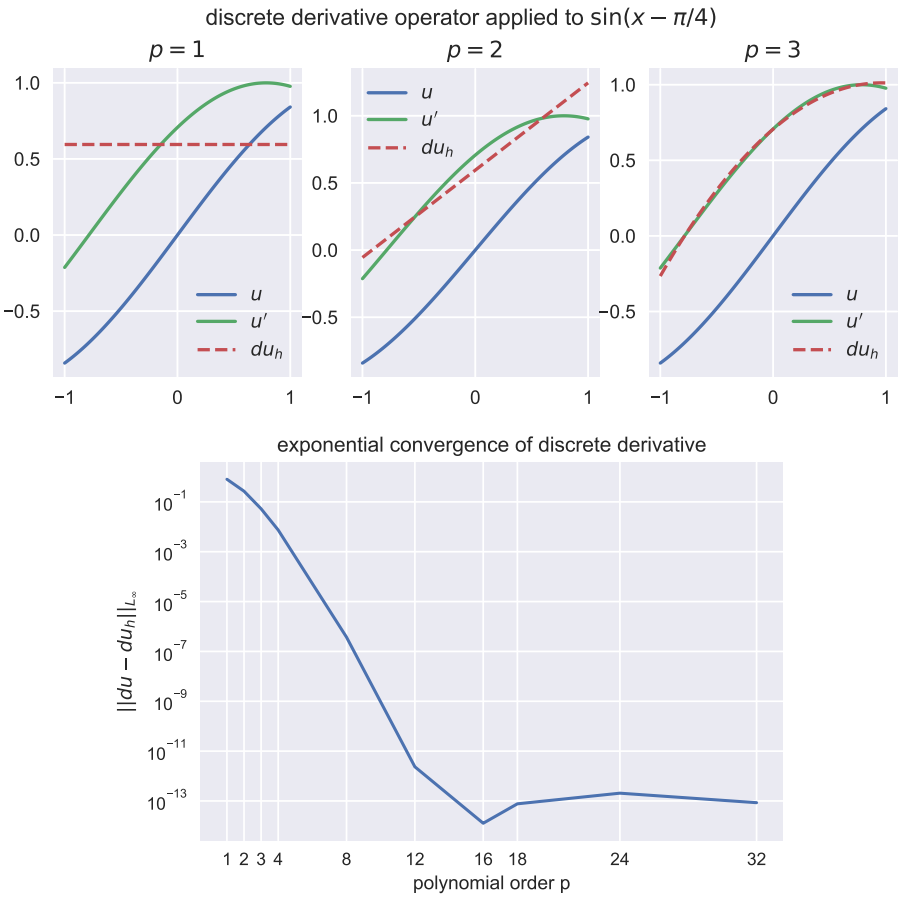


Figure 1-8: (Top) 1D discrete derivative operator applied to $\sin(x - \pi/4)$ on $[-1, 1]$. (Bottom) Exponential convergence of the discrete derivative.

transformation $\boldsymbol{\xi}(\boldsymbol{x})$, equation (1.48) can be used to assert that

$$\frac{\partial \phi}{\partial x} = \frac{\partial \phi}{\partial \xi}\frac{\partial \xi}{\partial x} + \frac{\partial \phi}{\partial \eta}\frac{\partial \eta}{\partial x}, \qquad \frac{\partial \phi}{\partial y} = \frac{\partial \phi}{\partial \xi}\frac{\partial \xi}{\partial y} + \frac{\partial \phi}{\partial \eta}\frac{\partial \eta}{\partial y}, \tag{1.80}$$

and so

$$\frac{\partial}{\partial x}(\cdot) = \frac{\partial \xi}{\partial x}\mathcal{D}_\xi + \frac{\partial \eta}{\partial x}, \mathcal{D}_\eta \qquad \frac{\partial}{\partial y}(\cdot) = \frac{\partial \xi}{\partial y}\mathcal{D}_\xi + \frac{\partial \eta}{\partial y}\mathcal{D}_\eta. \tag{1.81}$$

Now that we have defined these discrete operators, we can mix and match them in order to take gradients, divergences, curls, and so on, of the scalar or vector fields in question.

1.3 Numerical experiments

1.3.1 Verification

We verify the spatial convergence of our HDG implementation with the steady-state, diffusion-dominated problem similar to that presented in Nguyen [58]. We solve the model problem

Degree p	Mesh size h	$\|\|u_h - u\|\|_{L^2(\Omega)}$ Error	Order	$\|\|q_h - q\|\|_{L^2(\Omega)}$ Error	Order
1	1.00e+00	1.11e+00	-	4.54e+00	-
	5.00e-01	3.79e-01	1.55	1.21e+00	1.91
	2.50e-01	9.98e-02	1.93	3.05e-01	1.99
	1.25e-01	2.51e-02	1.99	7.64e-02	2.00
2	1.00e+00	1.44e-01	-	4.15e-01	-
	5.00e-01	2.03e-02	2.83	5.57e-02	2.90
	2.50e-01	2.57e-03	2.98	6.81e-03	3.03
	1.25e-01	3.21e-04	3.00	8.36e-04	3.03
3	1.00e+00	3.17e-02	-	9.95e-02	-
	5.00e-01	2.26e-03	3.81	6.57e-03	3.92
	2.50e-01	1.46e-04	3.95	4.19e-04	3.97
	1.25e-01	9.25e-06	3.99	2.63e-05	3.99

Table 1.1: Convergence history for the numerical quantities u_h and q_h to the exact solution in equation (1.82).

(1.1) on the domain $\Omega = (-1, 1) \times (-1, 1)$ with $\kappa = 1$. The source term f and Dirichlet boundary conditions g_D are chosen such that the exact solution is

$$u = \exp(x + y) \sin(\pi x) \sin(\pi y). \tag{1.82}$$

We consider a triangular mesh with uniform elements of size h covering Ω; we present the error and order of convergence in the L^2-norm in Table 1.1 for $p = 1$, 2, 3. Indeed we see the optimal order of convergence, $p+1$, in both the approximate scalar quantity u_h and its gradient q_h, consistent with the results in Nguyen [58].

1.3.2 Comparison between quadrature-free and quadrature-based integration

In section 1.2.5, we gave a cursory overview of the ideas behind the use of quadrature-free integral operators. We also presented results in section 1.3.1 in which we verified optimal convergence of the numerical solution with use of the discrete quadrature-free operators. A preliminary investigation of the accuracy of the quadrature-based and quadrature-free was conducted by Ueckermann and Lermusiaux [84], but the comparisons were limited to the context of a 1D discontinuous Galerkin scheme. In 1D, or indeed for any straight-sided 2D triangular element, the transform from master element to physical element is affine, and the approximation in (1.57) is exact. A better comparison of the accuracy between the two approaches can be made by examining the effect of a spatially varying $|J|$ over each element K.

As a numerical experiment, we consider the accuracy of numerical integration of a

smooth function which lies outside our polynomial approximation spaces defined in (1.7),

$$f(\boldsymbol{x}) = \sin(\pi x)\sin(\pi y), \tag{1.83}$$

over a region R (shown in Figure 1-9) discretized by non-affine 2D quadrilateral elements with Jacobian determinants $|J|$ that vary spatially over each element.

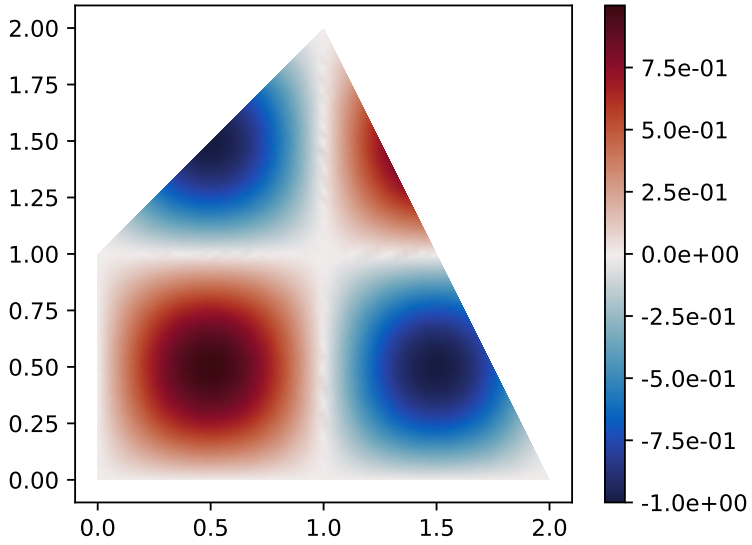


Figure 1-9: Function $f(\boldsymbol{x})$ (1.83) over the region R used in numerical test cases comparing quadrature-free and quadrature-based integral operators

We aim to investigate the accuracy differences between the two approaches in (1.84),

$$\begin{aligned}
\int_R f(\boldsymbol{x})\,d\boldsymbol{x} &= \sum_{K\in\mathcal{T}_h} \int_K f(\boldsymbol{x})\,d\boldsymbol{x} \\
&\approx \sum_{K\in\mathcal{T}_h} \widehat{\boldsymbol{\theta}_q}\,\mathrm{diag}\left(w_q \circ |J(\boldsymbol{x}_q)| \circ f(\boldsymbol{x}_q)\right)\widehat{\boldsymbol{\theta}_q}^T \qquad \text{(quadrature)} \\
&\approx \sum_{K\in\mathcal{T}_h} M\left[|J|_K \circ f(\boldsymbol{x}_n)\right] \qquad \text{(quadrature-free)},
\end{aligned} \tag{1.84}$$

as a function of both grid size h and polynomial order p. All quadrature rules are from Cools [19], and the same number of quadrature and nodal points are used to provide a fair comparison. All errors are reported in the L^2-norm.

First, we discretize R using a single finite element and examine the convergence of the numerical integrals (1.84) as a function of the polynomial order p. The purpose of such a test is to examine the differences in error magnitude over a single element as a function of polynomial order only. The results are shown in Figure 1-10.

The convergence behavior has several interesting features. The first observation is that both numerical integrals converge super-linearly in p and have roughly the same convergence behavior. We note that evaluating the quadrature-based integrals involves $\mathcal{O}(n_q^3) \sim 10^6$ operations over the course of the matrix multiplications at order $p = 10$, and therefore the roundoff error could be on the order of 10^{-10}; this is substantiated by the numerical results,

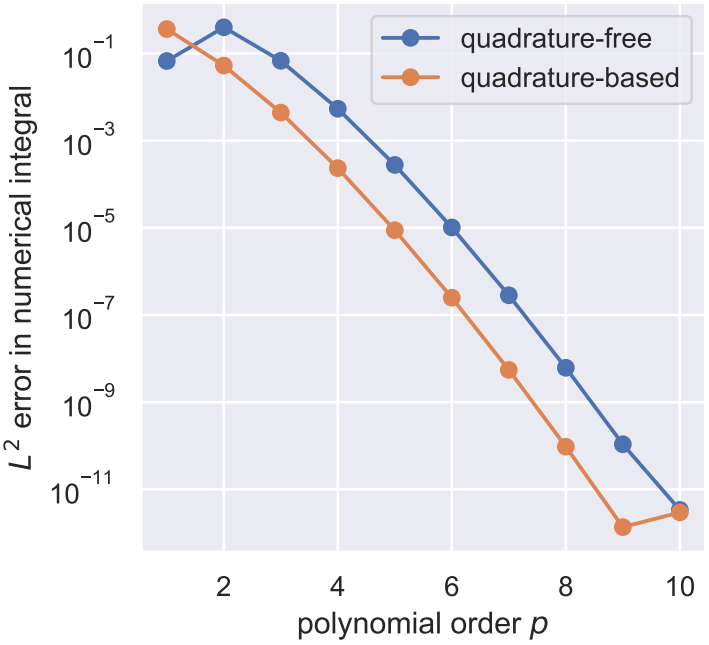


Figure 1-10: Comparison of L^2 error between quadrature-based and quadrature-free numerical integrals $\int_R f(\boldsymbol{x})\, d\boldsymbol{x}$ over a single finite element as a function of polynomial order p.

and we note that convergence to machine precision occurs at an absolute error of order 10^{-11}.

Another interesting feature is that the actual error magnitude of the quadrature-free integration appears to lag the quadrature-based error by an order of p. That is, the quadrature-free numerical integration on an order $p+1$ element appears to roughly have the same error as a quadrature-based integration on an order p element.

Lastly, while the convergence behavior of the quadrature-free integration is comparable to the quadrature-based approach, there is clearly a price paid for the ability to use a single, stored mass matrix M and to avoid interpolation of f to the quadrature nodes in terms of the absolute error of the numerical integral.

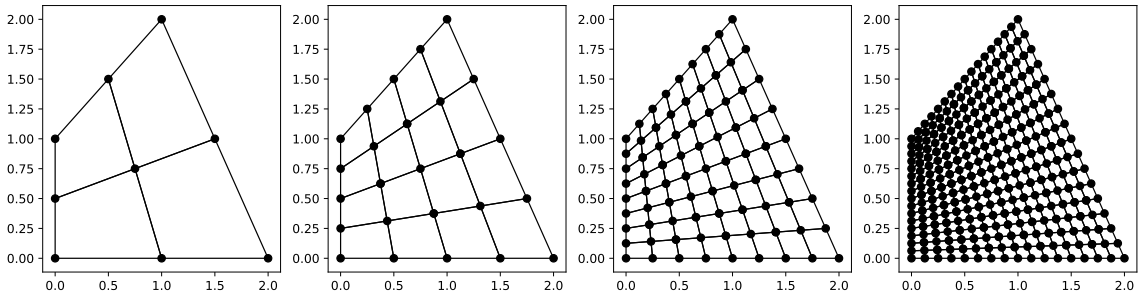


Figure 1-11: Sequence of non-affine quadrilateral meshes used to examine convergence

Degree p	Mesh size h	QF L^2 error	order	QB L^2 error	order
1	1.20e+0	6.75e-02	-	3.66e-01	-
	6.00e-01	1.30e-01	-0.95	1.39e-02	4.72
	3.00e-01	3.88e-02	1.74	4.87e-04	4.83
	1.50e-01	9.98e-03	1.96	2.74e-05	4.15
2	1.20e+0	4.01e-01	-	5.31e-02	-
	6.00e-01	1.93e-02	4.38	5.35e-04	6.63
	3.00e-01	7.26e-04	4.73	4.37e-06	6.94
	1.50e-01	4.11e-05	4.14	6.05e-08	6.17
3	1.20e+0	6.83e-02	-	4.42e-03	-
	6.00e-01	7.05e-04	6.60	1.17e-05	8.56
	3.00e-01	5.81e-06	6.92	2.23e-08	9.04
	1.50e-01	8.06e-08	6.17	7.62e-11	8.19
4	1.20e+0	5.42e-03	-	2.35e-04	-
	6.00e-01	1.46e-05	8.54	1.66e-07	10.47
	3.00e-01	2.79e-08	9.03	7.49e-11	11.11
	1.50e-01	9.52e-11	8.19	6.10e-14	10.26
5	1.20e+0	2.78e-04	-	8.83e-06	-
	6.00e-01	1.98e-07	10.45	1.64e-09	12.39
	3.00e-01	8.98e-11	11.11	1.75e-13	13.20
	1.50e-01	6.76e-14	10.38	7.47e-15	4.55

Table 1.2: Convergence history of the numerical integral $\int_R f(\boldsymbol{x})\, d\boldsymbol{x}$ as measured in the L^2-norm for quadrature-free (QF) and quadrature-based (QB) numerical integration over sequentially-refined computational meshes of representative element size h and at polynomial order p.

Next, in order to examine the convergence behavior of the numerical integral under grid refinement, we discretize the domain into successively-refined, non-affine quadrilateral meshes shown in Figure 1-11 and examine convergence of the numerical integrals (1.84) as a function of the polynomial order p and element size h. The convergence history in the L^2-norm is presented in Table 1.2.

We see that the convergence rates are higher for the quadrature-based schemes, as we would expect. We note that the convergence rates for the quadrature-based integrations are roughly order $2p+2$ where p is the polynomial order of the elements in the mesh, while the quadrature-free convergence rates are roughly $2p$. The suboptimal convergence rate at order $p = 5$ in the quadrature-based case is due to abutting machine precision as discussed in the single-element test case. We make the observation that the convergence rate of the quadrature free integrations $2p \geq p+1$ when $p \geq 1$, an observation which accounts for why we see optimal convergence at order $p+1$ in section 1.3.1 without using quadrature.

1.3.3 Errors incurred by isoparametric transformation

One of the advantages of the isoparametric mapping is the ability to handle non-affine and curved mesh geometries. Recall that the definition (1.37) asserts that the transformation from the nodal points on the master element to the nodal points on the physical element (the so-called nodal "interpolation points") is represented as a member of the polynomial space in which the finite element solution is sought. Therefore, if the actual transformation from master element to finite element lies outside the polynomial approximation space, there will be an error incurred due to the transformation approximation.

Non-affine, straight-sided meshes

In the case of two-dimensional mixed meshes (meshes containing triangular and quadrilateral elements), explicit criteria for eliminating transformation errors can be stated. Note that the transformation from the master triangular element to any physical element can always be written as a linear transformation. Similarly, the transformations from master edge to each physical space edge is also a linear transformation. Lastly, it can be readily shown that the transformation from the master quadrilateral to an arbitrary quadrilateral can be written as a quadratic transformation.

To illustrate this assertion, consider the model problem (1.1) with forcing f and inhomogeneous Dirichlet boundary conditions g_D chosen such that the exact solution is $u = x$ over the computational domain, with different choices of mesh. Since the solution $u = x$ is linear, we note that the solution is a member of any polynomial approximation space of order $p \geq 1$.

Figure 1-12 shows the numerical solution computed on an affine mesh using a polynomial approximation space of order $p = 1$. Since both physical space elements are affine transformations of the master element, a $p = 1$ transformation is sufficient to eliminate all transformation error; indeed, we see that the numerical solution u_h is correct to machine precision.

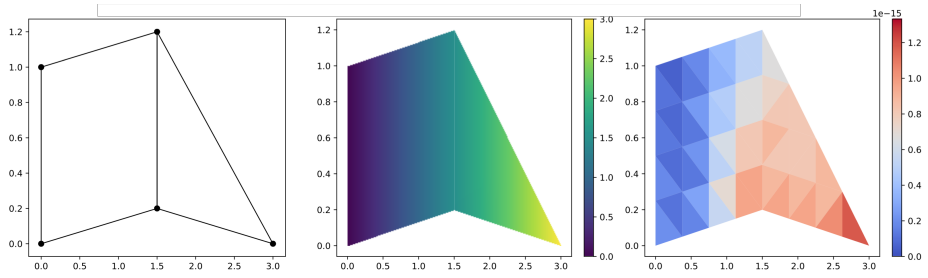


Figure 1-12: Numerical solution u_h (center) at order $p = 1$ of the manufactured solution $u = x$ on an affine mesh (left) and the difference $|u_h - u|$ (right).

A non-affine element is an element for which the mapping from the master element can not be expressed with a linear transformation. If we introduce a non-affine element into the computational mesh and solve the same problem, as shown in Figure 1-13, the errors are not machine precision. This is because the transformation from master quadrilateral to physical space quadrilateral is no longer representable in an approximation space of order $p = 1$. This is manifested in the nodal error, which is no longer machine precision—this departure from the exact solution is attributable to representation error in the isoparametric mapping,

namely, in the expression of the quadrature-free integral operators over the element interior of the single mesh quadrilateral.

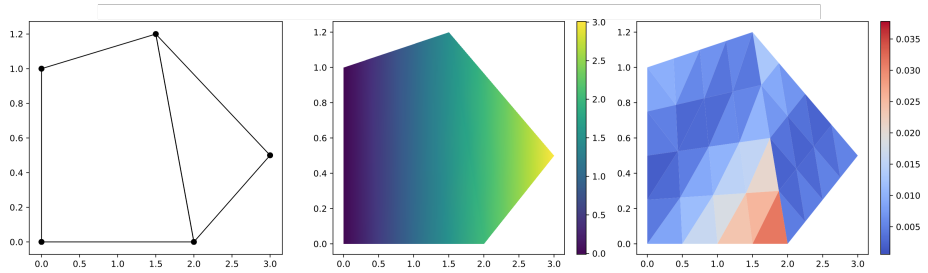


Figure 1-13: Numerical solution u_h (center) at order $p = 1$ of the manufactured solution $u = x$ on a non-affine mesh (left) and the difference $|u_h - u|$ (right).

Consider the same computational mesh as in Figure 1-13, but seeking a numerical solution u_h at order $p = 2$. The results are shown in Figure 1-14. Since the transformation is exactly representable in the space of quadratic functions, the transformation error is eliminated and once again u_h agrees with the exact solution to machine precision.

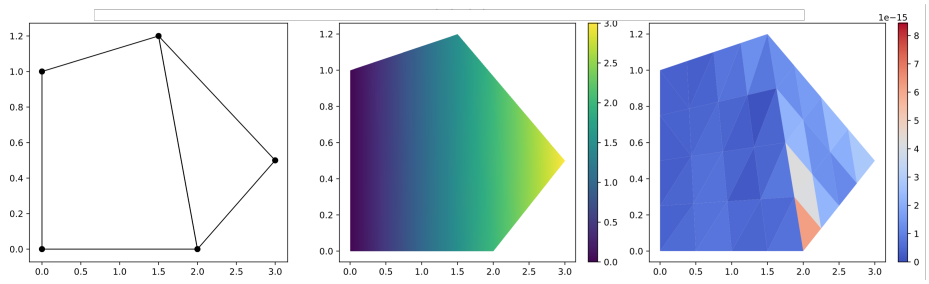


Figure 1-14: Numerical solution u_h (center) at order $p = 2$ of the manufactured solution $u = x$ on an affine mesh (left) and the difference $|u_h - u|$ (right).

Therefore, we claim that for straight-sided two-dimensional meshes, as long as the solution is sought in a polynomial approximation space of $p \geq 2$, there will not be error incurred due to the isoparametric mapping from master element to physical elements. The same conclusion applies to two-dimensional edge integrals in three-dimensional meshes.

For three-dimensional problems, we can produce similar conclusions. It can be shown that the mapping from a non-affine, straight-sided hexahedral element to the master hexahedral element is generally cubic. Therefore an isoparametric mapping at polynomial order $p \geq 3$ will be exact, otherwise an error due to the transformation will be incurred.

Curved meshes

For the case of arbitrary curved meshes, it is clearly not possible to make explicit assertions as to the polynomial order necessary to eliminate transformation error, as in the previous section. However, it is possible to examine the errors and convergence rates associated with curved boundaries.

As a test case, we consider solving the model problem on a circular domain. Mesh generators in general provide a straight-sided approximation to the domain of interest, but

this limitation can be overcome if we perturb the nodal degrees of freedom to coincide with the curved boundary of interest, as in Figure 1-15.

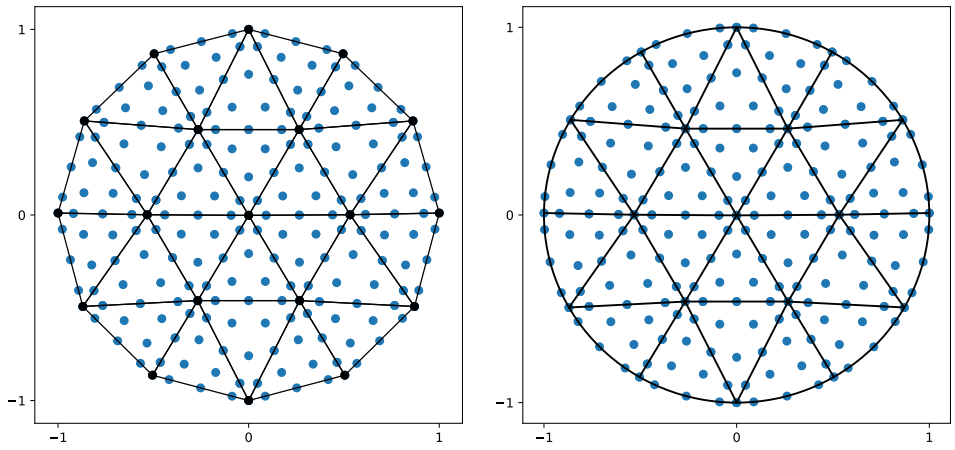


Figure 1-15: (Left) Before and (Right) after perturbation of nodal degrees of freedom to lie on curved, circular boundary.

Doing so has changed our interpolation points for each of the boundary elements, and the isoparametric mapping above will represent the transformation using the best polynomial approximation to the exact mapping (in this case, order $p = 4$).

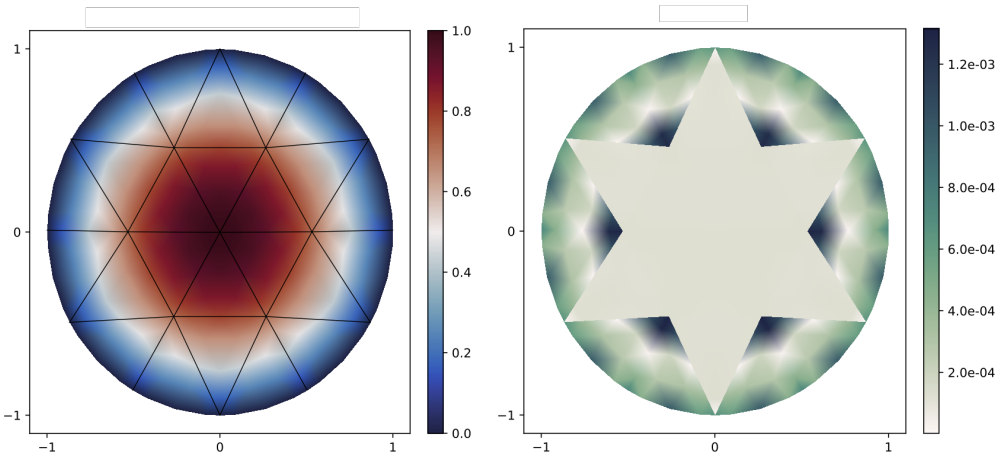


Figure 1-16: Numerical solution (left) for the manufactured solution $u_h = 1 - x^2 - y^2$ at order $p = 2$, and the difference from the exact solution $|u_h - u|$ (right)

Consider the simple manufactured solution $u_h = 1 - x^2 - y^2$ to the model problem posed on the unit circle. Note that since the solution lies in the polynomial space, we would expect the finite element solution to coincide with the analytical solution to machine precision. The plots of the numerical solution and that of the error are given in Figure 1-16.

Note that the numerical solution is not equal to its analytical counterpart to machine precision—there is error incurred by the representation of the curved boundary. The solution error is dominated by the error over the boundary elements, but the interior elements are affected as well. This is the price of representing curved geometries. However, the

magnitudes of the errors themselves are not meaningful; the more important question is how the convergence rates are affected.

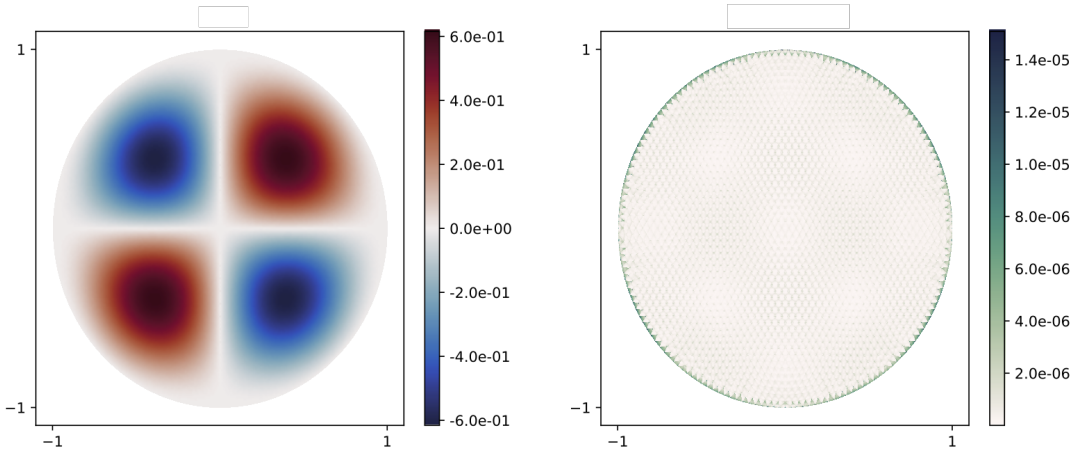


Figure 1-17: Numerical solution u_h at $p = 3$ (left) and $|u_h - u|$ (right).

To examine convergence rates, we solve the model problem (1.1) on a set of increasingly refined curved circular meshes at different polynomial orders; namely, $p = 1, 2, 3$. The forcing f and boundary conditions g_D are chosen such that the exact solution is

$$u = (1 - x^2 - y^2)\sin(\pi x)\sin(\pi y) \tag{1.85}$$

The numerical solution u_h at $p = 3$ on a circular mesh with mean element diameter $h = 0.04$ is shown in Figure 1-17, along with the difference $|u_h - u|$ over the computational domain. We note that even on such a refined mesh, the error is greatest around the boundary where the curved elements are present. The history of convergence is shown in Figure 1-18, where we see that the optimal rate of convergence in u_h is demonstrated.

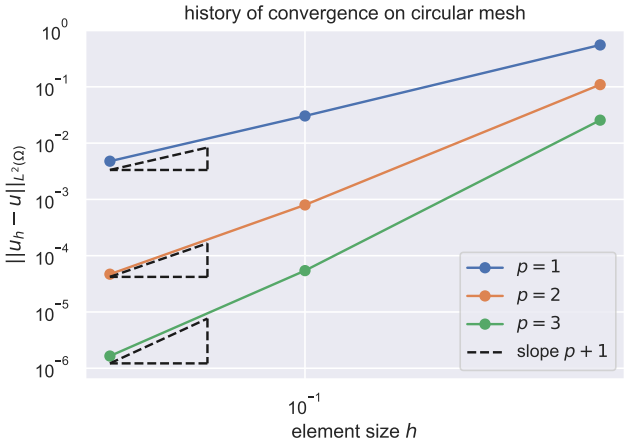


Figure 1-18: Convergence history of u_h in the L^2-norm for the manufactured solution (1.85) on a curved, circular mesh.

1.4 Summary

In this chapter we introduced the HDG methodology; we introduced a model problem which will provide an easily-generalizable basis for more complicated problems we will later encounter. We discussed the finite element spaces in which solutions to the model problem are sought, and their consequences in terms of boundary condition implementation. We characterized the problem for the global flux $\hat{u}_h$ and proposed a weak formulation which eliminates Dirichlet boundary condition information from the set of global unknowns, modifying the work done in [58], and guaranteeing symmetry of the nonzero entries in the global linear system.

Given the rich set of discretization choices, we summarized the classic block matrix procedure for the assembly of the global linear system and element-wise reconstruction specific to the HDG approach [58, 59], which will have important computational implications when dealing with matrix-free and parallel approaches to solving large problems. We described choices of discrete integral operators in both a quadrature-free and quadrature-based manner and discussed the implementation of an isoparametric mapping for handling curved elements and edges. We presented results for straight-sided, non-affine elements, verifying the polynomial orders above which no error will be incurred due to the isoparametric representation between master and physical element. This is, to our knowledge, not discussed in the literature; examination of isoparametric elements is typically focused on curvilinear boundaries [91]. However, the non-affine, straight sided cases are important for quadrilateral elements in 2D and for the wedge and hexahedral elements in 3D extruded meshes.

We concluded the chapter with a set of numerical experiments. We verified our implementation by demonstrating the theoretical optimal convergence behavior [13, 16], and reproducing results from [58]. Our new results presented a convergence study comparing quadrature-free and quadrature-based implementations of the discrete integral operators, which explained how we were able to obtain the optimal convergence rates without using quadrature.

Chapter 2

Efficient Computing for HDG Schemes

In this chapter, we more closely consider performance and efficiency with respect to the HDG methodology reviewed in Chapter 1. "Performance" and "efficiency" broadly describe measures of computational resources necessary to achieve a desired output, but are not sufficiently precise terms. For example, the interpretation of "efficiency" will differ dramatically when considering numerical software deployed on a low-power embedded system versus on a cloud-based platform consisting of thousands of compute nodes. Therefore, we provide context for our use of these terms by making the following presumptions:

1. We wish to solve a partial differential equation (PDE) at a certain resolution using an HDG discretization.
2. In our solving the PDE, we presume that time-to-solution is the primary limitation. A more "efficient" implementation will solve the same problem, with the same computational resources and accuracy, and subject to the same computational constraints in a shorter wall-clock time.
3. There exist computational constraints which must be obeyed, such as available memory, disk space, or available compute nodes.

Loosely speaking, our goal is to minimize time-to-solution on a single machine or in a distributed computing environment. Our discussion will necessarily include analysis of algorithmic performance and complexity, but we will also give particular attention to the design of efficient numerical code, as both are important with respect to time-to-solution and memory constraints. In particular, we follow the approach of starting with a naive implementation and considering first algorithmic improvements, followed by code-level (loops, vectorization, broadcasting, caching, and so on) improvements.

Modern benchmarking and profiling tools make it feasible to discriminate between different algorithms and implementations in a quantitative manner; however, the complexity of the HDG methodology as well as that of modern computer architectures is such that a single choice of algorithm is almost never sufficient to provide an efficient implementation at varying problem sizes. Hence, central to our discussion will be the identification of the different regimes at which various algorithms are applicable and performant.

In this chapter, we will begin by considering the naive, serial algorithm introduced in Chapter 1, paying particular attention to the efficient formation and application of the requisite discrete operators. We will outline efficient ways to vectorize the embarrassingly

parallel parts of the HDG algorithm, which will be applicable to any problem size. We will then turn our attention to the solution of the global linear system, conducting a scalability investigation to identify the memory and time-to-solution bottlenecks. We will discuss the applicability and implementation of direct and iterative solvers. Finally, we will introduce and analyze new matrix-free iterative solvers for HDG schemes, which merge the assembly and solution of the linear system.

2.1 Benchmarking test case

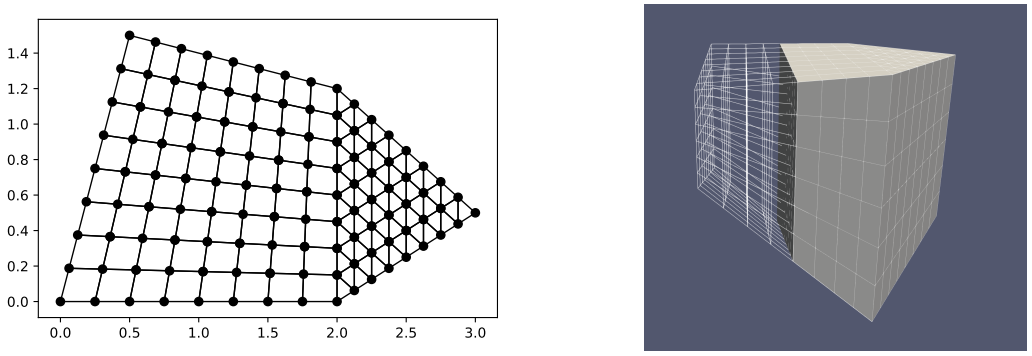


Figure 2-1: Top-down (left) and bathymetric view (right) of the 3D test case mesh.

We choose as a representative performance test case a straight-sided 3D test case with non-trivial bathymetry. The computational mesh can be seen in Figure 2-1; the mesh is an extrusion of the 2D mixed mesh with 6 terrain-following layers in the vertical direction. The mesh consists of 768 elements (384 hexahedral elements and 384 wedge elements). Since we noted in section 1.3.3 of Chapter 1 that $p \geq 3$ is needed to avoid errors in the numerical solution incurred due to a non-affine isoparametric transformation for 3D hexahedral elements, we benchmark using a numerical solution of order $p = 4$; there are 76,800 interior degrees of freedom and 68,320 edge degrees of freedom.

2.2 Serial algorithm

As a starting point, we modify equation (1.23) by negating the last two equations:

$$\begin{bmatrix} A & B & -C \\ B^T & -D & E \\ -C^T & G & -H \end{bmatrix} \begin{bmatrix} Q \\ U \\ \Lambda \end{bmatrix} = \begin{bmatrix} R \\ -F \\ -L \end{bmatrix}. \tag{2.1}$$

Recall that in section 1.2.6, we argued that a quadrature-free discretization of the integral operators breaks the symmetry of the element-local contributions and hence the symmetry of the left hand side $\mathbb{K}$ in the global problem $\mathbb{K}\Lambda = \mathbb{F}$. However, in the interest of generality, we would prefer to write the problem such that if a quadrature-based discretization is used,

symmetry is preserved. The elemental contributions to $\mathbb{K}$ and $\mathbb{F}$ are

$$\begin{aligned}\mathbb{K} &= -H - \begin{bmatrix}-C^T & G\end{bmatrix}\begin{bmatrix}A & B\\ B^T & -D\end{bmatrix}^{-1}\begin{bmatrix}-C\\ E\end{bmatrix},\\ \mathbb{F} &= L - \begin{bmatrix}-C^T & G\end{bmatrix}\begin{bmatrix}A & B\\ B^T & -D\end{bmatrix}^{-1}\begin{bmatrix}R\\ -F\end{bmatrix}.\end{aligned} \tag{2.2}$$

Importantly, the matrix

$$A_{loc} = \begin{bmatrix}A & B\\ B^T & -D\end{bmatrix} \tag{2.3}$$

is now analytically symmetric, and will be discretely symmetric if quadrature is used, which also produces symmetric contribution matrices $\mathbb{K}$.

Considering the modified system, we begin from the statement of the naive, matrix-based HDG algorithm stated in Algorithm 3. When available memory is not a constraint,

Algorithm 3 HDG Algorithm: serial, matrix-based

1: **for** $K \in \mathcal{T}_h$
2: compute elemental contributions $\mathbb{K}^K$
3: compute elemental contributions $\mathbb{F}^K$
4: **end for**
5: $\mathbb{K}$, $\mathbb{F} \leftarrow$ assemble $\mathbb{K}^K$, $\mathbb{F}^K$ for all $K \in \mathcal{T}_h$
6: $\Lambda \leftarrow$ solve $\mathbb{K}\Lambda = \mathbb{F}$
7: $\widehat{U} \leftarrow \Lambda \cup g_D$
8: **for** $K \in \mathcal{T}_h$
9: reconstruct $\boldsymbol{q}_h^K$, $u_h^K \leftarrow \begin{bmatrix}A & B\\ B^T & -D\end{bmatrix}^{-1}\left(\begin{bmatrix}0\\ -F\end{bmatrix} - \begin{bmatrix}-C\\ E\end{bmatrix}\widehat{u}_h^K\right)$.
10: **end for**

we can compute and store the element-local inverses:

$$A_{loc}^{-1} = \begin{bmatrix}A & B\\ B^T & -D\end{bmatrix}^{-1}, \tag{2.4}$$

which are used to compute both $\mathbb{K}$ and $\mathbb{F}$. The computation of $\mathbb{K}$, $\mathbb{F}$, and the element-wise reconstruction of Q and U (steps 3, 2, and 9, respectively) all involve the application of A_{loc}^{-1}, and storage and application of these arrays has important consequences for the scalability and efficient solution of the global linear system (step 6), and its handling will be discussed in detail in section 2.4.4.

2.3 Vectorization of element-wise operations

The formation of the elemental contributions $\mathbb{K}^K$ and $\mathbb{F}^K$ (steps 2 and 3 of Algorithm **??**, respectively), as well as the element-wise reconstruction of $\boldsymbol{q}_h$ and u_h (step 9) can all be completed independently and are embarrassingly parallelizable portions of the HDG algorithm. This section relates to the latter two — the efficient formation and assembly of the right-hand-side vector $\mathbb{F}$ and the elemental reconstruction of u_h and $\boldsymbol{q}_h$. Formation and assembly of $\mathbb{K}^K$ will depend on the method used to solve the global linear system, so we

defer its discussion to section 2.4. Since we are considering element-wise operations, these techniques are general and applicable to any problem size and are independent of how the global linear system is handled.

2.3.1 Elemental reconstruction of $\mathbf{q}_h$ and u_h

$$\begin{bmatrix} Q \\ U \end{bmatrix} = \begin{bmatrix} A & B \\ B^T & -D \end{bmatrix}^{-1} \left(\begin{bmatrix} 0 \\ -F \end{bmatrix} - \begin{bmatrix} -C \\ E \end{bmatrix} \widehat{u}_h \right). \tag{2.5}$$

$$\begin{bmatrix} C \\ E \end{bmatrix} \widehat{u}_h = \begin{bmatrix} \mathsf{L}\,\mathrm{diag}(|J_e| \circ n_k) M_e \\ \mathsf{L}\,\mathrm{diag}(|J_e| \circ \tau) M_e \end{bmatrix} \widehat{u}_h = \mathsf{L} \begin{bmatrix} \mathrm{diag}(|J_e| \circ n_k) \\ \mathrm{diag}(|J_e| \circ \tau) \end{bmatrix} M_e \widehat{u}_h \tag{2.6}$$

Since $\widehat{u}_h$ is a vector on each element edge space, we stand to gain by evaluating each of the operators sequentially on $\widehat{u}_h$ from right to left rather than from left to right as written since matrix-matrix multiplication is more expensive than matrix-vector multiplication. The vector $\widehat{u}_h$ has no exploitable structure, and in general is dense.

While applying the diagonal matrices to the product $M_e \widehat{u}_h$ is mathematically correct, forming and computing the matrix-vector product with the diagonal matrices is enormously inefficient, since most of the operations will involve multiplication by zero. Much better performance is achieved my broadcasting the vectors $M_e \widehat{u}_h$ directly onto the vectors $|J_e| \circ n_k$ and $|J_e| \circ \tau$:

$$\begin{bmatrix} \mathrm{diag}(|J_e| \circ n_k) \\ \mathrm{diag}(|J_e| \circ \tau) \end{bmatrix} [M_e \widehat{u}_h] = \begin{bmatrix} |J_e| \circ n_k \circ [M_e \widehat{u}_h] \\ |J_e| \circ \tau \circ [M_e \widehat{u}_h] \end{bmatrix} \tag{2.7}$$

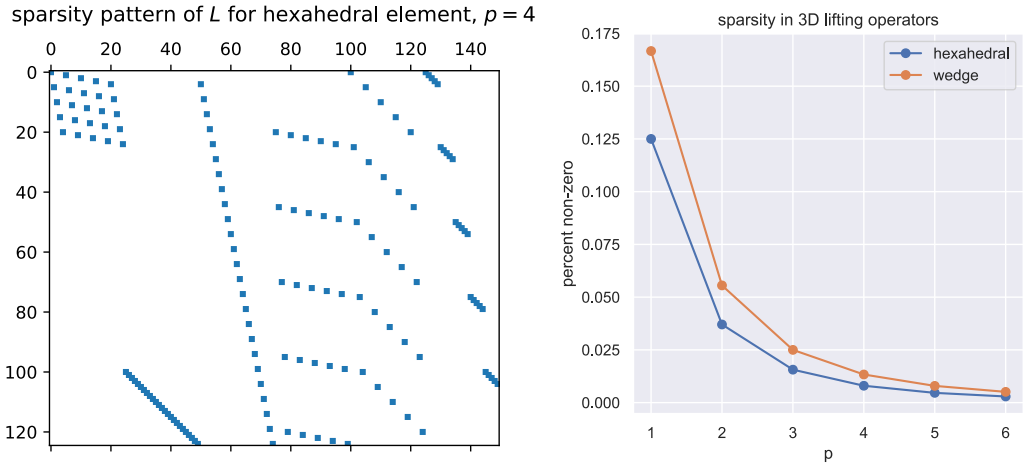


Figure 2-2: (Left) Sparsity pattern of lifting operator for a 3D hexahedral element at order $p = 4$, (Right) sparsity in 3D lifting operators as a function of polynomial order p.

Finally, application of the lifting operator L as a dense array is inefficient for the same reason; generally, 3D finite element lifting operators at polynomial order $p \geq 3$ tend to be sparse. Therefore, a sparse representation of L can be applied much more efficiently: for

example, the hexahedral lifting operator shown in Figure 2-2 can be applied with roughly a factor of 25 speedup (10.9 ms $\pm$ 273 μs vs 355 μs $\pm$ 9.12 μs) for our representative problem.

2.4 Solution of the global linear system $\mathbb{K}\boldsymbol{\Lambda} = \mathbb{F}$

All discussion heretofore has treated the solution of the global linear system $\mathbb{K}\boldsymbol{\Lambda} = \mathbb{F}$ as a black box which solves the globally-coupled weak problem in equation expressed in equation (1.15) for the numerical fluxes λ_h. In this section, we turn our attention to the scalability and details of forming and solving this linear system.

2.4.1 Direct methods

Direct methods are a good choice when the problem to be solved is small enough that a factorization of the linear system can be formed and stored in memory[1]. In such cases, it is generally advantageous to reduce bandwidth of the global linear system with a Cuthill–McKee reordering [51] (see, for example, Figure 2-3) and to store an LU factorization of the problem for fast solution of the global linear system $\mathbb{K}\Lambda = \mathbb{F}$ during time-stepping.

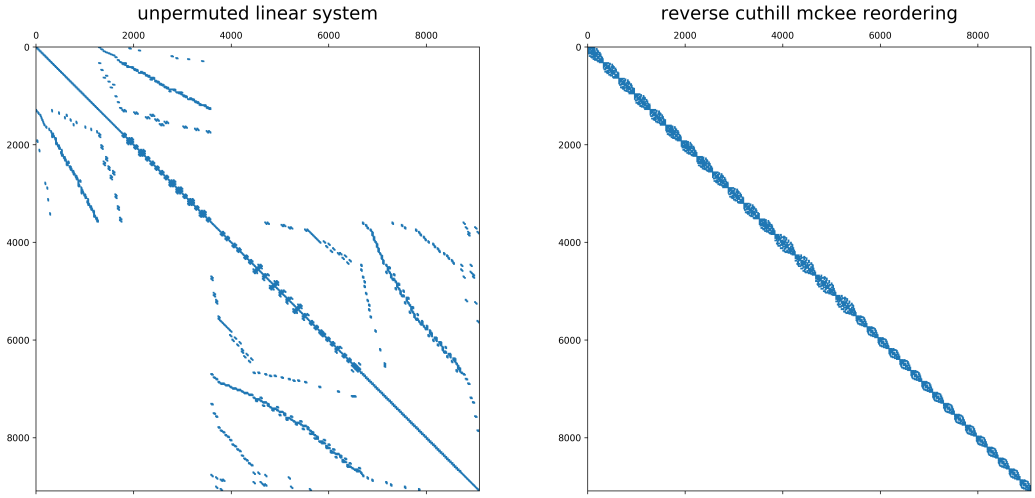


Figure 2-3: Bandwidth reduction of $\mathbb{K}$ for a 3D problem with the reverse Cuthill–McKee reordering [51], illustrated here for the model problem discretized on the benchmarking mesh in section 2.1.

Limitations of direct methods

While factorizations and direct solution methods are generally fast, robust, and usually the best choice for small problems [81], they require $O(n^3)$ work where n is the number of problem unknowns, and without careful attention do not preserve the sparsity that arises from discretization of integral or differential equations.

[1] We are presuming a static computational mesh in time. If the mesh itself is time-dependent, as in the case of a moving or adaptively refined mesh, then the expensive factorization of the linear system will simply be discarded at the next time step. In these cases, iterative approaches or direct elimination are better techniques.

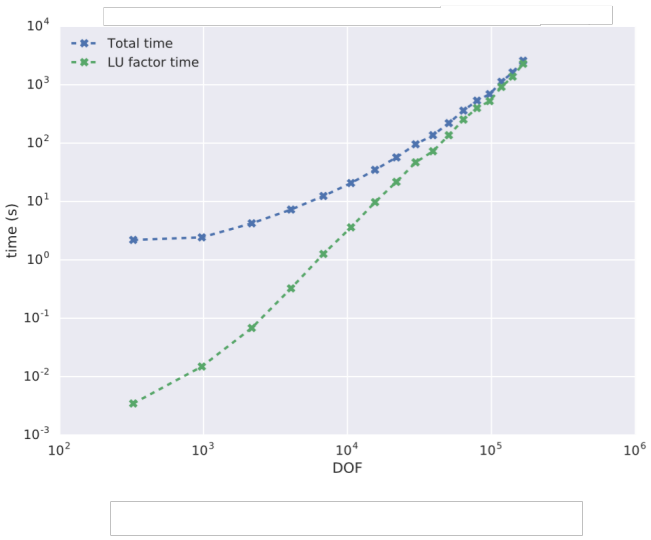


Figure 2-4: Comparison of LU factorization time against total wall-clock runtime when solving a time-dependent advection-diffusion problem, taking 100 time steps on a compute node with 16 of RAM

Figure 2-4 compares the LU factorization and total wall-clock time for a time-dependent (100 time steps) scalar advection-diffusion problem solved with an HDG discretization at different resolutions on a compute node with 16 GB of RAM. In this particular example, the resolution was increased until the underlying LU factorization (with Cuthill–McKee reordering) could no longer fit in main memory. Moreover, as problem size grows, the initial factorization time begins to dominate the time-to-solution. While this is a specific example, it illustrates the scalability limitations of direct solution techniques; even at a modest 10^5 unknowns, the factorization time and memory requirements become unacceptable.

2.4.2 Iterative methods

Iterative methods should be considered for large sparse problems for which excessive memory requirements become a bottleneck and a factorization or direct solution of the linear system $\mathbb{K}\Lambda = \mathbb{F}$ is no longer possible. Either memory requirements (factorization of $\mathbb{K}$ will require $O(n^2)$ memory) or time-to-solution requirements (factorization or elimination of $\mathbb{K}$ will require $O(n^3)$ operations) may be the limiting factor. In this work, we will consider Krylov subspace methods [81, 49], which treat the matrix $A \in \mathbb{R}^{n\times n}$ associated with a linear system as an operator and build up a subspace $\mathcal{K}_r(A, \boldsymbol{x}) = \text{span}\left\{\boldsymbol{x}, A\boldsymbol{x}, A^2\boldsymbol{x}, \ldots, A^{r-1}\boldsymbol{x}\right\}$ for $\boldsymbol{x} \in \mathbb{R}^n$ in relation to which an iterative solution is sought; for a complete treatment, see Saad [75]. The important feature of these methods from a computational standpoint is that Krylov subspace methods treat $\mathbb{K}$ as an operator and use repeated matrix-vector products $\mathbb{K}\boldsymbol{x}$ to iteratively construct the solution Λ, accurate to a given tolerance. The work required to compute $\mathbb{K}\boldsymbol{x}$ is proportional to the number of non-zeros in $\mathbb{K}$— $O(n)$ at each iteration. Therefore, use of iterative methods is essential to scale to large problem sizes.

Our primary concern in this chapter will involve the efficient application of $\mathbb{K}$ to a vector, since the matrix-vector product involving $\mathbb{K}$ is the limiting factor in Krylov subspace methods constructing the solution at the i^{th} iteration Λ_i. Our discussion will be generally

applicable to any Krylov subspace method, and we do not concern ourselves with the details of the Krylov methods themselves; these can readily be found in Golub [30] or Saad [75]. However, we will conclude the discussion on symmetry in section 1.2.6, which noted that if a quadrature-based scheme is used, or the domain geometry is sufficiently regular, the linear system $\mathbb{K}$ will be symmetric positive definite (SPD). In such cases, a conjugate gradient method will generally be the preferred Krylov subspace method. Otherwise, another choice of iterative solver is required; GMRES and BiCGSTAB are often suitable choices [73].

2.4.3 Matrix-free iterative methods

A matrix-free method refers to a general class of algorithms which apply the action of a matrix A on a vector $\boldsymbol{x}$ without explicitly storing or assembling the matrix. Rather, the elements of the matrix are computed and applied to $\boldsymbol{x}$ without being stored. Matrix-free methods can refer to explicit evolution schemes as well as to matrix-vector multiplication $A\boldsymbol{x}_i$ in the context of an iterative solve of an implicit problem; in this work, we refer to the latter. Matrix-free implicit methods are motivated by a lack of operative memory[2]—at certain problem sizes, the entire linear system can not be stored, necessitating a matrix-free approach.

In typical serial, matrix-based finite element algorithms such as Algorithm **??**, the assembly and solution of the global linear system are separate. In a matrix-free approach, we couple the two, computing the action of $\mathbb{K}$ on a vector $\boldsymbol{x}$ in order to form the iterative solution Λ_i.

Although matrix-free methods for HDG schemes require less memory, they generally require more computational time and specialized implementation, as will be discussed later in the chapter. The implementation details underlying matrix-free methods for HDG problems discussed herein are applicable to their use both on a single machine and in a parallel, distributed computing environment. The extension of the matrix-free methods to a distributed computing environment is outside the scope of this work, but is discussed in Foucart et al. [28].

Limitations of iterative methods

Iterative methods scale well as compared to direct methods, but it is important to identify and address the computational limiting factors both in memory and in time. In continuous Galerkin or standard implicit discontinuous Galerkin finite element schemes, these limitations are immediately apparent: storage and solution of the linear system are the memory and time-to-solution bottlenecks. However, for HDG schemes, the static condensation procedure in which $\boldsymbol{q}_h$ and u_h are eliminated to form a linear system for λ_h complicates scalability.

Equation (1.26) implies that that even if the linear system is assembled and stored in memory, computing the right–hand side $\mathbb{F}$ involves the forming the matrix inverse

$$A_{loc}^{-1} = \begin{bmatrix} A & B \\ -B^T & D \end{bmatrix}^{-1} \tag{2.8}$$

[2] Sometimes it is also the case that the underlying linear system is simple enough (e.g., a tri-diagonal linear system) that applying the operator to the vector directly in a matrix-free sense can be more efficient and faster than storing and applying a sparse data structure. However, this is not the case in an HDG finite element method.

or its factorization for every element. The inverse A_{loc}^{-1} represents the HDG parametrization of the element-local PDE onto the edge space, and is distinct from the element-local contributions to a standard CG or DG finite element scheme. Somewhat surprisingly, the application of this inverse is an important computational consideration for large HDG problems.

To see this, we consider the scaling of memory requirements of the HDG data structures for an $N \times N$ square domain in 2D, and an $N \times N \times N$ cubic domain in 3D at different polynomial orders p. We consider a mesh consisting entirely of quadrilateral and hexahedral elements, respectively. The regularity of the domain allows exact calculation of the memory required for all of the relevant data structures. As a reference, we investigate the scaling assuming a single, large compute node with 256 GB of main memory, and we make the assumption that data is stored in double-precision floating point format, with 64 bits required for each floating point number.

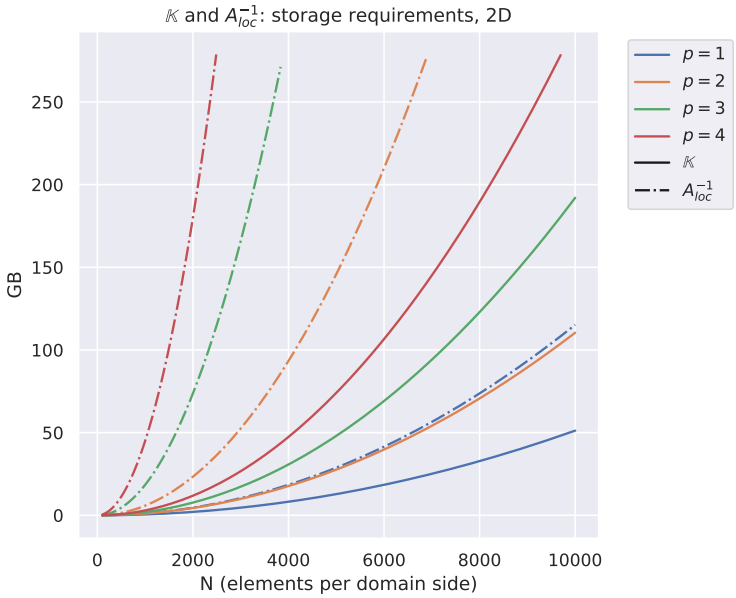


Figure 2-5: Memory requirements for $\mathbb{K}$ and A_{loc}^{-1} for an $N \times N$ 2D square domain (quadrilateral elements) at different polynomial orders p.

Figure 2-5 shows a comparison of the memory requirements for the assembled linear system $\mathbb{K}$ compared to the dense A_{loc}^{-1} arrays. We see that the A_{loc}^{-1} arrays rather than the linear system $\mathbb{K}$ are the memory bottleneck, and the effect becomes even more dramatic as polynomial order increases. This is because the inverses are generally dense, and scale as $n_K \left[(d+1)n_b\right]^d$ where n_b is number of degrees of freedom on each element, n_K is number of elements, and d is problem dimension. By comparison, $\mathbb{K}$ scales with the number of edges in the mesh rather than elements, and the degrees of freedom on each edge are of dimension $d-1$. We remark that the storage requirements of A_{loc}^{-1} also involve the duplicated spatial degrees of freedom common to DG schemes. So from a memory standpoint, the A_{loc}^{-1} arrays are the largest data structure, and a memory-efficient way of applying the inverses is crucial as problem size and polynomial order grows. Techniques to address these issues will be discussed at length in section 2.4.4.

How do the memory requirements compare to the master-to-physical element transfor-

mation data J^{-1}, $|J|$, and $|J_e|$? Figure 2-6 includes the transformation data in the memory requirements. We see that the memory required to store the transformation data is modest compared to that of the linear system $\mathbb{K}$ and the A_{loc}^{-1} arrays.

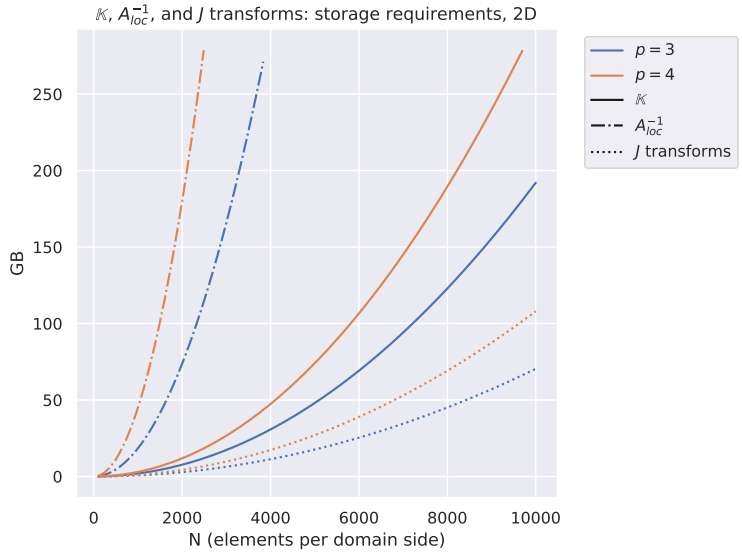


Figure 2-6: Memory requirement comparison between $\mathbb{K}$, A_{loc}^{-1}, and transformation data J^{-1}, $|J|$, and $|J_e|$ for an $N \times N$ 2D square domain at different polynomial orders p.

For 2D problems, if we explicitly store neither the global linear system nor the element local inverses, we can handle much larger problem sizes. In 3D we would expect the same conclusion to hold, but more dramatically.

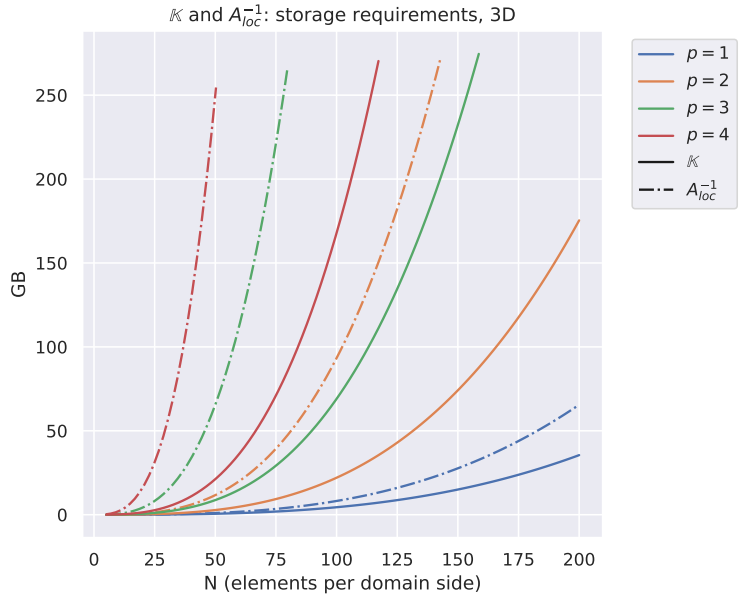


Figure 2-7: Memory requirements for $\mathbb{K}$ and A_{loc}^{-1} for an $N \times N \times N$ 3D cubic domain (hexahedral elements) at different polynomial orders p.

Figure 2-7 shows that this is indeed the case. The 2D trends are exacerbated, and we see that without storing the A_{loc}^{-1} arrays, the linear system alone limits problem size to roughly $N = 100$ elements per side at order $p = 4$ on a 256 GB compute node. Furthermore, storing the element local inverse arrays limits the problem size to fewer than $N = 40$ elements per side! In summary, the primary concern with regards to memory limitations is the storage or computation of the A_{loc}^{-1} arrays. If these are not stored, then storage of the sparse linear system $\mathbb{K}$ becomes the bottleneck.

Matrix-free methods are aimed at removing the memory bottleneck of $\mathbb{K}$, but applying $\mathbb{K}$ implicitly requires applying A_{loc}^{-1} — if the inverse is not available, applying $\mathbb{K}$ would require inversion of each A_{loc} matrix at every iteration of every iterative solve at every time step. If we were able to efficiently apply A_{loc}^{-1} without needing to store the array, we would have only the master-to-physical element mapping data J^{-1}, $|J|$, $|J_e|$ and the normal vectors as the comparatively modest memory requirements, and we could devise an efficient matrix-free implementation for the HDG schemes addressed earlier in this work. This is the primary motivation for the chapter.

2.4.4 Efficient Application of A_{loc}^{-1}

In order to form the elemental contribution matrices, it is required to explicitly invert the element local system, or to solve a system of linear equations each time an elemental contribution is required. This represents the parametrization of the element-local PDE onto the edge space. As we saw in section 2.4.3, the storage and application of A_{loc}^{-1} has important implications for iterative and matrix-free schemes. To motivate the following procedure, we will discuss some of the properties of the element local system.

Observe that in both the strong and weak DG forms of the model problem, equations (1.28) and (1.8), respectively, the inner product $(\boldsymbol{q}_h, \boldsymbol{v})_K$ has no differential operators applied to either $\boldsymbol{q}_h$ or $\boldsymbol{v}$. Using the quadrature-free integral operators discussed at length in section 1.2.5, the A_{loc} block matrix A consists of the discretization $A^k = \text{diag}\left(\left[\kappa^{-1}\right]^k \circ |J|_K\right) M$, where k is the coordinate direction, and M is the mass matrix over the master element $\widehat{K}$. This discretization admits the following manipulation:

$$
\begin{aligned}
A_{loc} &= I \begin{bmatrix} A & B \\ B^T & -D \end{bmatrix} \\
&= \begin{bmatrix} M & & \\ & M & \\ & & M \end{bmatrix} \begin{bmatrix} M^{-1} & & \\ & M^{-1} & \\ & & M^{-1} \end{bmatrix} \begin{bmatrix} A_x & & B_x \\ & A_y & B_y \\ B_x^T & B_y^T & -D \end{bmatrix} \\
&= \begin{bmatrix} M & & \\ & M & \\ & & M \end{bmatrix} \begin{bmatrix} |J|\kappa_x^{-1} I & & M^{-1}B_x \\ & |J|\kappa_y^{-1} I & M^{-1}B_y \\ M^{-1}B_x^T & M^{-1}B_y^T & -M^{-1}D \end{bmatrix},
\end{aligned}
\tag{2.9}
$$

where the extension to 3D problems is immediately apparent. With this manipulation, the portion of A_{loc} corresponding to the A block matrix reduces to a diagonal matrix with respect to the unknown Q, as shown in Figure 2-8.

The diagonal block immediately suggests an exploitable structure in terms of computa-

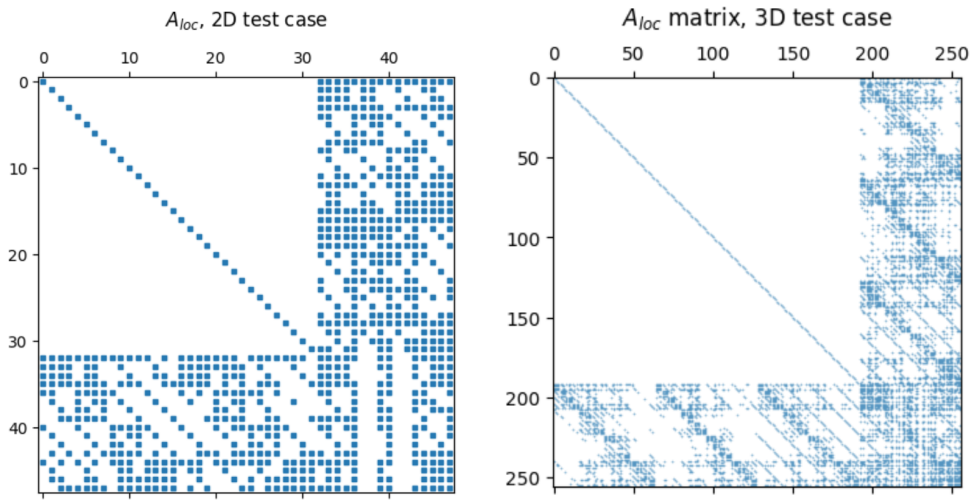


Figure 2-8: Exploitable sparsity patterns for 2D and 3D A_{loc} matrices (divided by the elemental mass matrix) described in equation (2.9).

tional efficiency. The inverse matrix A_{loc}^{-1} is

$$\begin{aligned} A_{loc}^{-1} &= \left(\begin{bmatrix} M & & \\ & M & \\ & & M \end{bmatrix} \begin{bmatrix} |J|\kappa_x^{-1} I & & M^{-1}B_x \\ & |J|\kappa_y^{-1} I & M^{-1}B_y \\ M^{-1}B_x^T & M^{-1}B_y^T & -M^{-1}D \end{bmatrix} \right)^{-1} \\ &= \begin{bmatrix} |J|\kappa_x^{-1} I & & M^{-1}B_x \\ & |J|\kappa_y^{-1} I & M^{-1}B_y \\ M^{-1}B_x^T & M^{-1}B_y^T & -M^{-1}D \end{bmatrix}^{-1} \begin{bmatrix} M^{-1} & & \\ & M^{-1} & \\ & & M^{-1} \end{bmatrix}. \end{aligned} \tag{2.10}$$

Importantly, M^{-1} is the inverse mass matrix on the master element, which is pre-computed and stored. Therefore, we turn our attention to the other matrix in equation (2.10), which we can denote as

$$(M^{-1}A_{loc})^{-1} = \begin{bmatrix} |J|\kappa^{-1} I & M^{-1}B \\ M^{-1}B^T & -M^{-1}D \end{bmatrix}^{-1}, \tag{2.11}$$

where $M^{-1}B$ is an abuse of notation, but should be clear from equation (2.10). The inverse of a matrix

$$\Theta = \begin{bmatrix} A & B \\ B^T & D \end{bmatrix} \tag{2.12}$$

can be written

$$\Theta^{-1} = \begin{bmatrix} A^{-1} + A^{-1}BS^{-1}B^TA^{-1} & -A^{-1}BS^{-1} \\ -S^{-1}B^TA^{-1} & S^{-1} \end{bmatrix}, \tag{2.13}$$

where $S = D - B^TA^{-1}B$ is the Schur complement of Θ. This form is convenient, because in the case of $(M^{-1}A_{loc})^{-1}$, A^{-1} is diagonal and trivial to compute algebraically:

$(|J|\kappa^{-1}I)^{-1} = (\kappa/|J|)I$. Hence we can write the inverse $(M^{-1}A_{loc}^{-1})$ as

$$(M^{-1}A_{loc})^{-1} = \begin{bmatrix} (|J|^{-1}\kappa I) + (|J|^{-1}\kappa I)B\,\mathbb{S}^{-1}B^T(|J|^{-1}\kappa I) & -(|J|^{-1}\kappa I)B\,\mathbb{S}^{-1} \\ -\mathbb{S}^{-1}B^T(|J|^{-1}\kappa I) & \mathbb{S}^{-1} \end{bmatrix}, \quad (2.14)$$

where $\mathbb{S} = -D - B^T(|J|^{-1}\kappa I)B$.

Storing the complete, fully-dense inverse of A_{loc}^{-1} would be wasteful. Instead, we can compute and store the Schur complements $\mathbb{S}^{-1}$ on each element, which are the same size as D. Then, for example, applying A_{loc}^{-1} to the right–hand side vector $[R, -F]$, as in equation (2.2) amounts to computing

$$\mathbb{F} = L - \begin{bmatrix} -C^T & G \end{bmatrix} \begin{bmatrix} \left[(|J|^{-1}\kappa I) + (|J|^{-1}\kappa I)B\,\mathbb{S}^{-1}B^T(|J|^{-1}\kappa I)\right] M^{-1}R + (\kappa I)B\,\mathbb{S}^{-1}M^{-1}F \\ -\mathbb{S}^{-1}B^T(|J|^{-1}\kappa I)M^{-1}R - \mathbb{S}^{-1}M^{-1}F \end{bmatrix}, \quad (2.15)$$

which consists of application of small, pre-computed matrix operators. Computing the products, we are careful to both evaluate all chained matrix products right to left, since repeated matrix-vector multiplication is more efficient than repeated matrix-matrix multiplication; and to apply $(|J|^{-1}\kappa I)$ operators directly via broadcasting— it's wasteful to perform matrix multiplication with a dense matrix consisting almost entirely of zero entries.

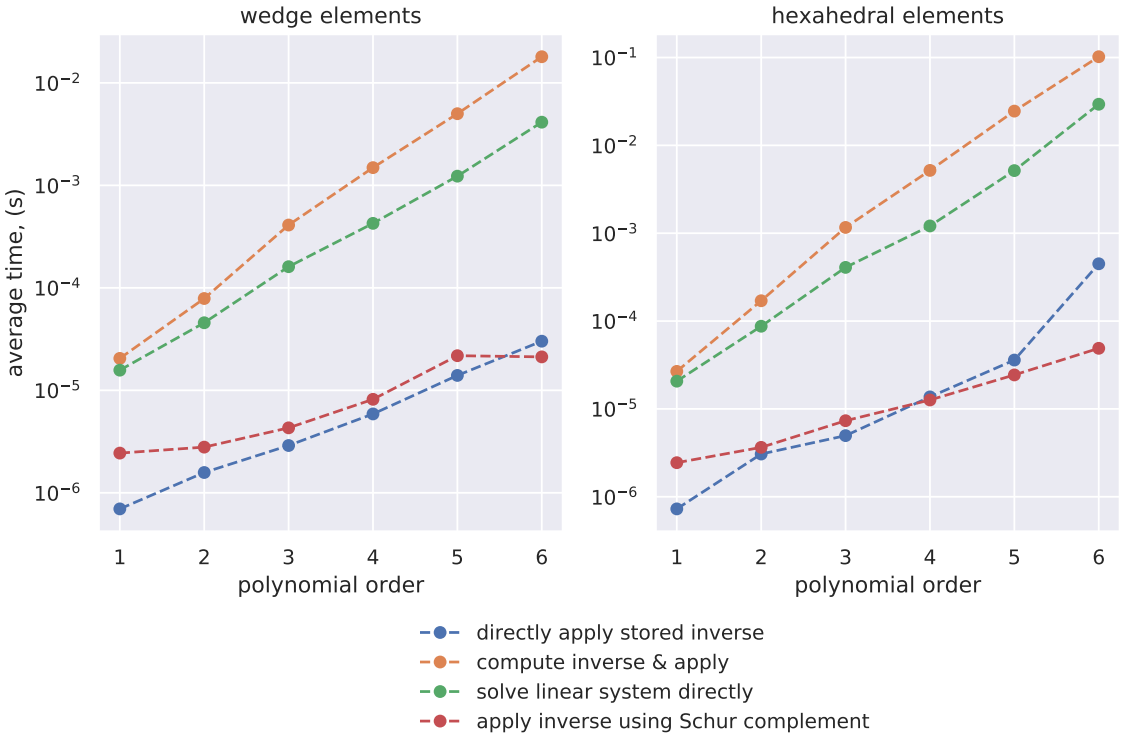


Figure 2-9: Benchmarking data for application of A_{loc}^{-1} to a random vector at different polynomial orders for 3D elements. Average wall-clock time out of 10,000 trials using a single representative wedge or hexahedral element; standard deviations were negligible compared to mean times in all cases.

Figure 2-9 shows benchmarking data for different approaches to applying A_{loc}^{-1}. Take as a reference the time required to directly apply (in a vectorized way) a stored inverse

or factorization; we showed in section 2.4.3 that for large problems, storing the dense A_{loc}^{-1} inverses becomes a memory bottleneck and is not feasible. If we are operating in the matrix-free regime, where the linear system can not be stored in memory, then the dense inverses certainly can not be stored either. Hence, we are left with the option of computing and applying A_{loc}^{-1}; solving the linear system

$$\begin{bmatrix} A & B \\ B^T & -D \end{bmatrix} \begin{bmatrix} Q \\ U \end{bmatrix} = \begin{bmatrix} -C \\ E \end{bmatrix} \tag{2.16}$$

and applying the matrix-vector product of $\mathbb{K}$ to a vector $\boldsymbol{x}$ as

$$\mathbb{K}\boldsymbol{x} = \left(-H - \begin{bmatrix} -C^T & G \end{bmatrix} \begin{bmatrix} Q \\ U \end{bmatrix} \right) \boldsymbol{x}, \tag{2.17}$$

or by storing the small, dense Schur complements $\mathbb{S}^{-1}$ (or their factorizations) and applying them as in equation (2.15) to reproduce the action of A_{loc}^{-1}. Unsurprisingly, benchmarking shows directly computing and applying A_{loc}^{-1} is by far the slowest; while it requires nothing be stored, these inverses must be computed and applied for every element, at every iteration of the solution of the linear system $\mathbb{K}\Lambda = \mathbb{F}$. It is well-known in numerical linear algebra that it is almost never efficient to explicitly compute a matrix inverse, and indeed, we see that solving the linear system instead of applying the inverse is faster. Lastly, note that for hexahedral elements of polynomial orders $p \geq 3$ and for wedge elements of polynomial order $p \geq 5$, the Schur complement approach is even faster than direct application of a stored inverse for A_{loc}.[3]

How does storage of $\mathbb{S}^{-1}$ compare memory-wise to storage of the A_{loc}^{-1} arrays? We revisit the scaling of memory requirements for 3D problems shown in Figure 2-7. Figure 2-10 shows the memory requirements associated with storing the linear system $\mathbb{K}$ compared to the $\mathbb{S}^{-1}$ arrays. We see that the memory requirements associated with A_{loc}^{-1} have been substantially reduced — now storage of $\mathbb{K}$ is the bottleneck — and we still don't need to perform any inversions or factorizations during each matrix-free application of $\mathbb{K}$.

Returning for a moment to our investigation of the memory requirements associated with matrix-based and matrix-free iterative linear solvers in section 2.4.3, we saw that on a compute node with with 256 GB of RAM, A_{loc}^{-1} requirements limited a 3D problem at polynomial order $p = 4$ to fewer than $N = 40$ elements per side. However, Figure 2-10 shows that storing $\mathbb{S}^{-1}$ instead allows $N = 100$ elements per side, which constitutes a substantial improvement.

The Schur complement approach represents a good trade-off between wall-clock time and memory in the matrix-free context; but asymptotically, even the small $\mathbb{S}^{-1}$ matrices will become a memory bottleneck. However, the Schur complement approach is still useful since $\mathbb{S}$ will always be smaller than the A_{loc} matrix. Rather than defaulting to directly solving the A_{loc} linear system, the factorization of $\mathbb{S}$ should be computed on the fly and $\mathbb{S}^{-1}$ applied through back-substitution every time $\mathbb{S}^{-1}$ appears in the chain of operators when applying the operator $\mathbb{K}$.

[3]Figure 2-9 only shows up to order $p = 6$ to demonstrate the crossover point in application times, and benchmarking to higher order ($p = 12$) shows that the Schur complement inverse remains faster at high order, although asymptotically both inverse applications scale the same way. This is to be expected, since both are ultimately matrix-vector multiplications.

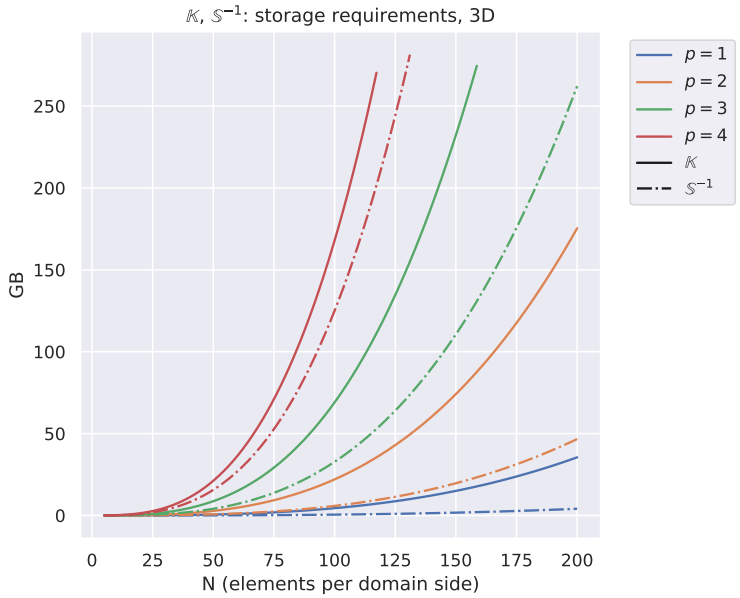


Figure 2-10: Memory requirements for $\mathbb{K}$ and $\mathbb{S}^{-1}$ for an $N \times N \times N$ 3D square domain (quadrilateral elements) at different polynomial orders p.

Stability considerations

The previous section illustrated the main ideas behind exploiting the structure of A_{loc} in order to apply its inverse, A_{loc}^{-1}, efficiently by storing only the small, dense inverse Schur complements $\mathbb{S}^{-1}$. While we saw the benefit of this approach in terms of both required memory and wall-clock time of inverse application, the approach as stated can suffer from numerical stability issues.

To illustrate this, we consider the solution of the system in equation (2.16), for a wedge element[4] in the test case introduced in section 2.1. Figure 2-11 shows the solution of equation (2.16) directly by Gauss elimination (to machine precision) compared to the solution $A_{loc}^{-1}\boldsymbol{b}$ applied using the inverse Schur complement $\mathbb{S}^{-1}$.

We see that while the qualitative behavior of the system is correct, the relative error of the solution is on the order of 10^{-3}, which is unacceptable from a numerical stability point of view. The issue is not due to the condition number of the Schur complement $\mathbb{S}$ —in fact, the $\mathbb{S}$ matrix is generally much better conditioned than the A_{loc} matrix. Closer examination shows the primary source of error to be roundoff error due to the relative magnitudes of the $A^{-1} = \kappa/|J|$ operator and the unscaled operands $[-C, E]$ or $[R, -F]$.

To resolve the relative magnitude issue, we can modify the procedure by multiplying the entire equation by $M^{-1}\,\mathrm{diag}(\kappa/|J|)$, yielding

$$\begin{aligned} A_{loc}^{-1} &= \left(\begin{bmatrix} \mathrm{diag}(|J|/\kappa)M & B \\ B^T & -D \end{bmatrix}\right)^{-1} \\ &= \begin{bmatrix} I & M^{-1}\,\mathrm{diag}(\kappa/|J|)B \\ M^{-1}\,\mathrm{diag}(\kappa/|J|)B^T & -M^{-1}\,\mathrm{diag}(\kappa/|J|)D \end{bmatrix}^{-1} \begin{bmatrix} M^{-1}\,\mathrm{diag}(\frac{\kappa}{|J|}) & \\ & M^{-1}\,\mathrm{diag}(\frac{\kappa}{|J|}) \end{bmatrix}. \end{aligned} \tag{2.18}$$

[4]The choice of element is arbitrary and purely illustrative, since the large relative error present in this element is present in the other elements as well.

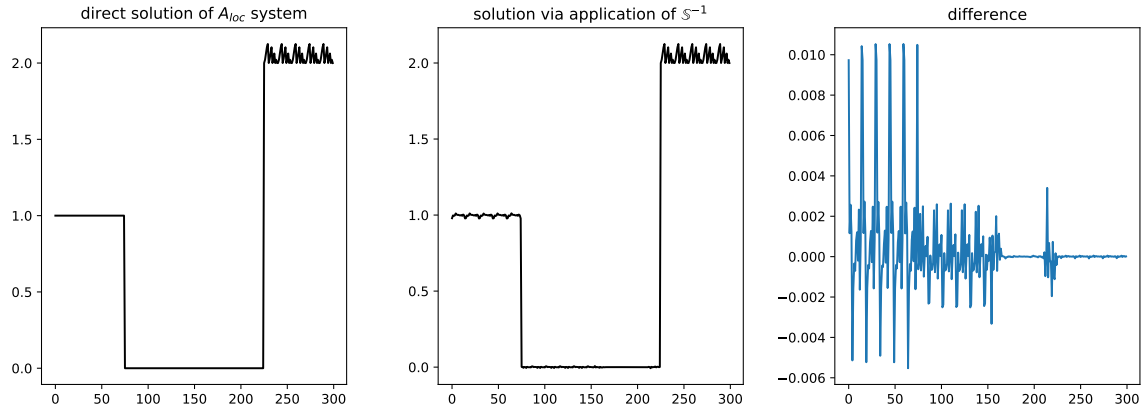


Figure 2-11: Comparison of direct solution of equation (2.16) with solution via application of the inverse Schur complement $\mathbb{S}^{-1}$.

This manipulation applies the $\kappa/|J|$ scaling to both sides of the A_{loc} system. It bears repeating that $\text{diag}(\kappa/|J|)$ is mathematical notation, but is computationally carried out as a element-wise product with a vector operand or row-scaling rather than a matrix-multiplication. The Schur complement becomes, since the upper left block matrix $A = A^{-1} = I$,

$$\begin{aligned} \tilde{D} &= M^{-1}\,\text{diag}(\kappa/|J|)D, \\ \tilde{B} &= M^{-1}\,\text{diag}(\kappa/|J|)B, \\ \tilde{\mathbb{S}} &= -\tilde{D} - \tilde{B}^T\tilde{B}, \end{aligned} \tag{2.19}$$

and the inverse becomes

$$\begin{bmatrix} I & \tilde{B} \\ \tilde{B}^T & -\tilde{D} \end{bmatrix}^{-1} = \begin{bmatrix} I + \tilde{B}\tilde{\mathbb{S}}^{-1}\tilde{B}^T & -\tilde{B}\tilde{\mathbb{S}}^{-1} \\ -\tilde{\mathbb{S}}^{-1}\tilde{B}^T & \tilde{\mathbb{S}}^{-1} \end{bmatrix}. \tag{2.20}$$

The operators $\tilde{C}$ and $\tilde{E}$, as well as vectors $\tilde{R}$ and $\tilde{F}$ are formed by pre-multiplication with $M^{-1}\,\text{diag}(\kappa/|J|)$ in the same way. This ameliorates the accumulation of roundoff error, because both equation (2.20) and the right hand side operand, e.g., $[\tilde{R}, \tilde{F}]$, are multiplied by the factor $\kappa/|J|$, which, large or small, will scale both operator and operand.

This modification makes the approach more numerically robust. Figure 2-12 shows the application of A_{loc}^{-1} via $\tilde{\mathbb{S}}^{-1}$ to the same system as in Figure 2-11; the results are much better and the relative error is on the order of 10^{-11}.

2.4.5 Alternative static condensation

We can make use of the previous two ideas (pre-multiplication by M^{-1} and storing the inverse Schur complement $\mathbb{S}^{-1}$) to derive an alternate matrix-free scheme which eliminates the system in equation (2.1) for Λ directly. The advantage of doing so is that in the evaluation of the matrix-vector product $\mathbb{K}\boldsymbol{x}$, there is no need to consider the A_{loc}^{-1} operator explicitly; rather, we apply the matrix-vector product directly as a chain of operators acting on $\boldsymbol{x}$.

We introduced the process of static condensation in section 1.1.1 by algebraically elim-

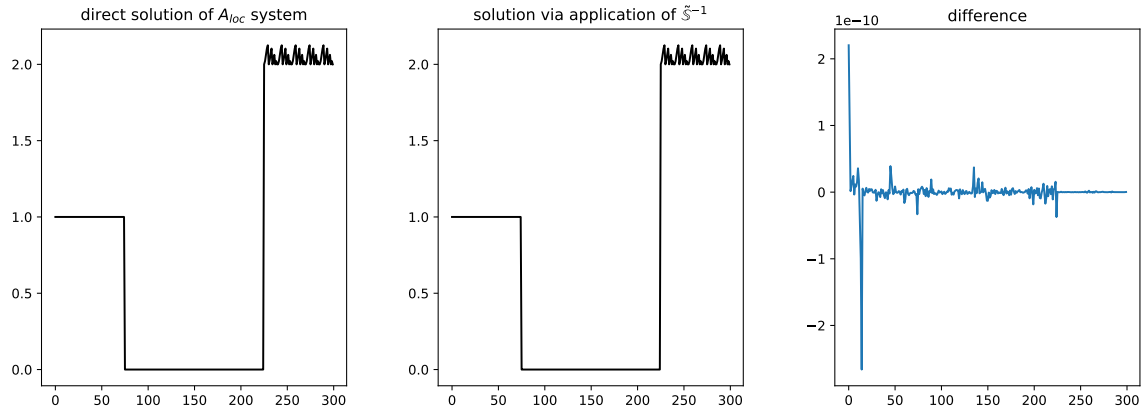


Figure 2-12: Comparison of direct solution of equation (2.16) with solution via application of the inverse Schur complement $\tilde{\mathbb{S}}^{-1}$.

inating equation (1.23) for the scalar trace Λ. However, the decision to eliminate the variables $[Q, U]$ was arbitrary. We could instead algebraically eliminate the matrix system for Λ directly to offer an alternative perfective.

We begin with equation (2.1), but where the entire system has been pre-multiplied by the inverse mass matrix M^{-1} on the master element:

$$\begin{bmatrix} A & B & -C \\ B^T & -D & E \\ -C^T & G & -H \end{bmatrix} \begin{bmatrix} Q \\ U \\ \Lambda \end{bmatrix} = \begin{bmatrix} R \\ -F \\ -L \end{bmatrix}, \tag{2.21}$$

where, for example, A refers to $|J|\kappa^{-1}I$. Elimination of the first equation yields $Q = A^{-1}(R - BU + C\Lambda)$. Substitution of Q and re-grouping yields the following reduced system for U, Λ:

$$\begin{bmatrix} -(D + B^T A^{-1} B) & B^T A^{-1} C + E \\ C^T A^{-1} B + G & -C^T A^{-1} C - H \end{bmatrix} \begin{bmatrix} U \\ \Lambda \end{bmatrix} = \begin{bmatrix} -F - B^T A^{-1} R \\ -L + C^T A^{-1} R \end{bmatrix}. \tag{2.22}$$

Once again, let $\mathbb{S} = -D - B^T A^{-1} B$. This system could be solved directly for U, Λ and Q could be found by substitution. However, this is undesirable, since one of the primary advantages of the HDG approach is to avoid the duplication of degrees of freedom present in U on each element edge in the global linear system. Instead, eliminate U as

$$U = \mathbb{S}^{-1}\left[-\left(F + B^T A^{-1} C\right) - \left(B^T A^{-1} C + E\right)\Lambda\right]. \tag{2.23}$$

Back-substitution and straightforward matrix manipulations yields

$$\begin{aligned} &-\left[\left(C^T A^{-1} B + G\right)\mathbb{S}^{-1}\left(B^T A^{-1} C + E\right) + \left(C^T A^{-1} - H\right)\right]\Lambda \\ &\quad = -L + C^T A^{-1} R + \left(C^T A^{-1} B + G\right)\mathbb{S}^{-1}\left(F + B^T A^{-1} R\right), \end{aligned} \tag{2.24}$$

yielding the explicit elemental contribution operators

$$\begin{aligned}\mathbb{K} &= \left(C^T A^{-1} B + G\right) \mathbb{S}^{-1} \left(B^T A^{-1} C + E\right) + \left(C^T A^{-1} - H\right) \\ \mathbb{F} &= -L + C^T A^{-1} R + \left(C^T A^{-1} B + G\right) \mathbb{S}^{-1} \left(F + B^T A^{-1} R\right).\end{aligned} \tag{2.25}$$

We emphasize that this approach is equivalent to that of the previous sections, since the Schur complement approach is a way to invert the A_{loc} system. However, instead of considering A_{loc}^{-1} as an operator to be applied, we can instead re-write the entire element-local contributions $\mathbb{K}^K$ and $\mathbb{F}^K$ in terms of the Schur complement explicitly —in this case we don't need to consider the A_{loc} system at all, but rather we simply chain the discrete matrix operators together to form the action of $\mathbb{K}$. This is a more convenient form for a matrix-free representation.

We remark that instead of explicitly computing the inverse $\mathbb{S}^{-1}$ directly, an appropriate factorization of $\mathbb{S}$ should be stored and the inverse applied to a vector $\boldsymbol{a}$ by solving $\mathbb{S}\boldsymbol{x} = \boldsymbol{a}$ by back-substitution. Writing $\mathbb{S}^{-1}$ is simply a mathematical convenience, but not an indication as to how the operator is actually applied. Lastly, in the regime where even the $\mathbb{S}^{-1}$ factorizations are too large to be stored, the inverse can be applied to a vector a in a numerically stable way by solving the linear system $\mathbb{S}\boldsymbol{x} = \boldsymbol{a}$ when evaluating the matrix-vector products $\mathbb{K}\boldsymbol{x}$ over the course of an iterative solve.

2.4.6 Matrix-free algorithm

Since we have addressed the memory bottleneck of applying A_{loc}^{-1}, we can now illustrate the procedure for computing an HDG solve in a matrix-free manner. The modifications to the matrix-based algorithm are given in Algorithm 4.

Algorithm 4 HDG Algorithm: serial, matrix-free

1: **for** $K \in \mathcal{T}_h$
2: compute elemental Schur complement inverse $\tilde{\mathbb{S}}^{-1}$
3: **end for**
4: $\mathbb{F} \leftarrow$ assemble $\mathbb{F}^K$ for all $K \in \mathcal{T}_h$
5: $\Lambda \leftarrow$ matrix-free iterative solve $\mathbb{K}\Lambda = \mathbb{F}$
6: $\widehat{U} \leftarrow \Lambda \cup g_D$
7: **for** $K \in \mathcal{T}_h$
8: reconstruct $u_h^K \leftarrow \tilde{\mathbb{S}}^{-1}\tilde{B}^T\tilde{C}\widehat{u}_h^K - \tilde{\mathbb{S}}^{-1}(\tilde{F} - \tilde{E}\widehat{u}_h^K)$
9: reconstruct $\boldsymbol{q}_h^K \leftarrow -(I + \tilde{B}\tilde{\mathbb{S}}^{-1}\tilde{B}^T)\tilde{C}\widehat{u}_h^K + \tilde{B}\tilde{\mathbb{S}}^{-1}(\tilde{F} - \tilde{E}\widehat{u}_h^K)$
10: **end for**

However, it remains to detail the procedure for the matrix-free iterative solve in Algorithm 4, step 5. Any choice of Krylov subspace method is applicable; here we use the conjugate gradient Krylov subspace method for the matrix-free iterative solve [75, 81], without loss of generality.

Algorithm 5 Standard conjugate gradient

1: $i \leftarrow 0$
2: $\boldsymbol{d}, \boldsymbol{r} \leftarrow (\mathbb{F} - \mathbb{K}\boldsymbol{\lambda}_0)$
3: $\delta_0, \delta_{new} \leftarrow \boldsymbol{r}^T\boldsymbol{r}$
4: **while** $i < i_{max}$ **and** $\delta_{new} > \varepsilon^2\delta_0$
5: $\quad \boldsymbol{q} \leftarrow \mathbb{K}\boldsymbol{d}$,
6: $\quad \alpha \leftarrow \frac{\delta_{new}}{\boldsymbol{d}^T\boldsymbol{q}}$
7: $\quad \boldsymbol{\lambda} \leftarrow \boldsymbol{\lambda} + \alpha\boldsymbol{d}$
8: $\quad$ **if** recompute_criterion(i) **then**
9: $\quad\quad \boldsymbol{r} \leftarrow (\mathbb{F} - \mathbb{K}\boldsymbol{\lambda})$,
10: $\quad$ **else**
11: $\quad\quad \boldsymbol{r} \leftarrow (\boldsymbol{r} - \alpha\boldsymbol{q})$
12: $\quad$ **end if**
13: $\quad \delta_{old} \leftarrow \delta_{new}$, $\delta_{new} \leftarrow \boldsymbol{r}^T\boldsymbol{r}$
14: $\quad \beta \leftarrow \frac{\delta_{new}}{\delta_{old}}$
15: $\quad \boldsymbol{d} \leftarrow (\boldsymbol{r} + \beta\boldsymbol{d})$
16: $\quad i \leftarrow (i + 1)$
17: **end while**

An efficient implementation of the conjugate gradient algorithm is given in Shewchuk [78] and reproduced in Algorithm 5. Steps 2, 5, and 9 involve the matrix-free application of the operation of $\mathbb{K}$ to $\boldsymbol{\lambda}_0$, $\boldsymbol{d}$, and $\boldsymbol{\lambda}$, respectively. These matrix-free matrix-vector applications are the most expensive part of the algorithm; therefore to decrease computational cost, step 8 only involves explicit computation of the residual vector $\boldsymbol{r}$ at certain iteration numbers, in order to remove accumulation of floating point errors. One such example is recomputing the residual explicitly every $\sqrt{n}$ iterations, where n is the size of the solution vector $\boldsymbol{\lambda}$ [78]. The algorithm for the matrix-free matrix-vector product is given in Algorithm 6.

Algorithm 6 Matrix-free matrix-vector product $\mathbb{K}\boldsymbol{\lambda}$

1: $\boldsymbol{y} \leftarrow \boldsymbol{0}$
2: **for** $K \in \mathcal{T}_h$
3: $\quad \mathcal{E} \leftarrow$ global DOF indices on $\partial K \setminus \Gamma_D$
4: $\quad \boldsymbol{\lambda}^K \leftarrow \boldsymbol{\lambda}[\mathcal{E}]$
5: $\quad \boldsymbol{y}[\mathcal{E}] \leftarrow \boldsymbol{y}[\mathcal{E}] - H\boldsymbol{\lambda}^K - \begin{bmatrix} -C^T & G \end{bmatrix} \begin{bmatrix} (I + \tilde{B}\tilde{\mathbb{S}}^{-1}\tilde{B}^T)\tilde{C}\boldsymbol{\lambda}^K + \tilde{B}\tilde{\mathbb{S}}^{-1}\tilde{E}\boldsymbol{\lambda}^K \\ -\tilde{\mathbb{S}}^{-1}\tilde{B}^T\tilde{C}\boldsymbol{\lambda}^K - \tilde{\mathbb{S}}^{-1}\tilde{E}\boldsymbol{\lambda}^K \end{bmatrix}$
6: **end for**
7: $\boldsymbol{\lambda} \leftarrow \boldsymbol{y}$

The vector $\boldsymbol{\lambda}$ contains all of the degrees of freedom on the element interfaces $\varepsilon \setminus \varepsilon_D^\partial$. The degrees of freedom on each interior edge $e \in \varepsilon^\circ$ will be operated on by $\mathbb{K}^{K^+}$ and $\mathbb{K}^{K^-}$ (using the $K^\pm$ notation as in section 1.1.1 to denote the elements on the left and right side of an edge), the elemental contributions from elements on either side of the interface (see Figure 1-2). As such, the application of $\mathbb{K}$ is not embarrassingly parallel — there is a race condition[5] updating the entries of $\boldsymbol{y}$. In a matrix-based procedure, this is avoided, summing the contributions of $\mathbb{K}^{K^+}$ and $\mathbb{K}^{K^-}$ during assembly. Moreover, the matrix-free

[5] A race condition occurs when two software processes must operate on the same memory, preventing the processes from doing so simultaneously.

algorithm can not be done in place; both $\mathbb{K}^{K+}$ and $\mathbb{K}^{K-}$ must operate on the original vector $\boldsymbol{\lambda}$, necessitating use of a destination vector $\boldsymbol{y}$.

Algorithms 4, 5, and 6 together constitute the complete algorithm for performing an HDG solve in a matrix-free manner, storing only the master-to-physical element transformation data, the inverse Schur complements $\tilde{S}^{-1}$, and the right-hand-side vector $\mathbb{F}$.

2.5 Summary

In this chapter, we addressed scalability and efficient implementation of the HDG schemes in Chapter 1. We provided vectorization techniques for the embarrassingly parallel portions of the HDG algorithm and were able to obtain significant time savings in their application; discussion of the embarrassingly parallel parts of the algorithm is not discussed in the literature, but we included relatively minor changes that can result in large speedups.

We discussed choices and applicability of direct and iterative solvers to the solution of the global linear system $\mathbb{K}\mathbf{\Lambda} = \mathbb{F}$, and found that handling of the A_{loc}^{-1} was crucial to scaling the HDG algorithm to large problems. Indeed, the memory requirements for these arrays grow much faster than the requirements for storing the global linear system due to the duplication of degrees of freedom which the HDG method seeks to mitigate in the first place. The cost of storing versus recomputing the local matrices A_{loc} is mentioned in Fabien et al. [27], but the authors state that the focus of the work is not on the generation or solution of the local matrices.

To address this issue, we presented the derivation of a novel, numerically stable approach to efficiently apply the A_{loc}^{-1} operators without having to explicitly store the arrays, taking advantage of the structure of the A_{loc} linear system and the form of the quadrature-free integral operators introduced in Chapter 1. Alternative techniques to address this issue have been the subject of very recent research [41, 42]; however, these techniques rely on Schur complement approaches for the augmented system for $[U, \Lambda]$ rather than only the numerical trace Λ, and use quadrature-based integral operators (sometimes with nodal points collocated with Gauss-Lobatto integration points). Our procedure is unique in that it exploits the quadrature-free representation of the integral operators to efficiently compute the Schur complement. Doing so approach addressed the memory bottleneck in the scalability investigation. We concluded by synthesizing all of these techniques to provide an efficient, quadrature-free, memory-friendly, matrix-free implementation of the HDG algorithm.

Chapter 3

Automated verification of HDG software

Maintaining correctness of scientific-computing codes is challenging, complex, and no single practice has been demonstrated to effectively prevent all errors; however, there is evidence that a combination of practices can be very effective in identifying and preventing mistakes when used in conjunction with one another [23, 87]. The widely-used open-source finite element libraries `deal.II` [7], DUNE [9, 38], and FEniCS [52] all employ an extensive set of automated tests in order to ensure software correctness.

Understandably, most test-driven development references such as Percival [65] or Govindaraj [31] concern general commercial application development and are thus too broad in scope to be applicable in their entirety to numerical codebases. However, useful types of automated testing from the broader field of software engineering directly applicable to numerical software include: unit testing, integration-to-main testing (integration testing), and regression testing. Unit testing refers to tests that test the functionality of a single function or "unit" of the codebase by establishing that the function returns the known, correct output for a known input (for example, a test that a factorial function returns 120 as the output to the input 5). Integration tests test the behavior of larger subsystems of code and the functionality of a code on known test cases (for example, ensuring a root-finding software package returns the correct roots to a non-linear equation system). Regression testing tests against failures that are caused by changes to the environment rather than by the developer (for example, code that fails due to an unresponsive compute node, or due to an updated external library file).

In this chapter, we focus on integration testing specific to verification of our HDG schemes. In order for the automated testing to be effective, we must keep several goals in mind. An ideal test suite is comprehensive and ensures that any change to the codebase does not break any existing code functionality. Otherwise, every change to the codebase must be accompanied by manually re-testing previously trusted test cases, which is time-consuming, error-prone, and ultimately unmaintainable as the codebase grows. However, since running numerical experiments can be expensive, we would like to design the integration tests such that they provide the most possible code coverage but are able to run quickly enough to not stall code development — for example, a test suite that takes several hours or days to run can be useful at times, but would seriously hinder code development if they had to be run several times per day; more frequent tests should ideally be able to run on the order of seconds to minutes. Lastly, we would like the integration tests to provide more information

than the breaking of previously functional code; we would like the tests to be hierarchical in a way that helps indicate the source of the break in functionality.

The contribution of this chapter is a procedure for verifying the "kernel" of an HDG software, which takes an input the mesh and boundary conditions, and outputs the solution variables $\boldsymbol{q}_h$ and u_h to the discretized problem for a chosen polynomial order p. We will demonstrate a method to provide automated, fast verification in a way that provides diagnostic information upon test failure. Extension of these tests to time-marching schemes is straightforward. Our approach is unique in that it is specific to HDG, can be used for 2D and 3D problems, and can be evaluated quickly. For all tests, we use the `pytest` framework [26, 62], but the verification techniques discussed herein are generally applicable to any HDG finite element software.

3.1 Strategies for finite element integration testing

3.1.1 Method of manufactured solutions

The method of manufactured solutions is a well-known technique in numerical methods — see, for example, Roache [71, 72] — and forms the basis for the automated integration tests considered in this chapter. For the model problem given in equation (1.1), the method of manufactured solutions involves choosing a solution u along with a diffusivity κ which may vary in space, and analytically evaluating the forcing function f as well as the Neumann boundary condition values g_N on Γ_N.

The process of analytically evaluating the quantities f and g_N can also be automated using symbolic manipulation software in a dimension-independent manner. All manufactured solutions in this work are computed using the `SymPy` library [53].

3.1.2 Convergence tests

Convergence tests are a commonly used verification method for finite element methods in which a numerical solution u_h is computed on successively refined grids along with the L^2-error. The rate of convergence r as specified by $\|u - u_h\|_{L^2(\Omega)} \leq Ch^r$, where C is a constant independent of the representative element size h, and can be computed and compared against the theoretical optimal convergence rate.

While convergence tests are necessary for verification of finite element software and are straightforward to automate, they are expensive; the numerical solution must be computed successively and with increasing resolution, along with the error $\|u - u_h\|_{L^2(\Omega))}$, which may involve quadrature. Moreover, the convergence rates are asymptotic — the numerical solution must be sufficiently well-resolved in order to observe the correct convergence order.

The cost of creating an automated convergence test can be mitigated by determining the asymptotic convergence regime *a posteriori*. Take, for example, the convergence results for the numerical experiment given in Table 1.1. We see from the table that the optimal rate of convergence is obtained only between the two finest mesh sizes. An automated integration test could be made less costly by computing the numerical solution only on these meshes and verifying the optimal rate of convergence. However, those meshes are already refined to the point that they contain a non-trivial number of elements.

3.1.3 Exact polynomial tests

A particularly useful subset of manufactured solutions for efficient verification of finite element software are exact polynomial tests — choosing a manufactured $u \in V_h^p$, that is, a solution that is an element of the approximation space in which the solution is sought. As a consequence, the numeric solution u_h and its gradient $\boldsymbol{q}_h$ should agree with the analytical solution u and $\kappa \nabla u$, respectively, up to machine precision. These tests are efficient because only one spatial solve is required, and unlike convergence tests, can be conducted on arbitrarily small meshes. Therefore, multiple exact polynomial tests can be conducted quickly and can be parameterized with differing testing features (boundary conditions, linear solvers, and so on).

3.2 HDG integration test hierarchy

While we saw that both convergence tests and exact polynomial tests can verify the numerical solutions returned by finite element software, upon failure, they cannot indicate the portion of the software responsible for the failure of the test. However, we can specify a hierarchy of increasingly complex test cases in order to isolate portions of the HDG algorithm responsible for the test case failure. The tests can be run sequentially or in parallel; it is by noting which test cases succeed and which tests fail that can illustrate the offending code.

Algorithm 7 HDG Algorithm

1: **for** $K \in \mathcal{T}_h$
2: compute elemental contributions $\mathbb{K}^K$, $\mathbb{F}^K$
3: **end for**
4: $\mathbb{K}$, $\mathbb{F} \leftarrow$ assemble $\mathbb{K}^K$, $\mathbb{F}^K$ for all $K \in \mathcal{T}_h$
5: $\Lambda \leftarrow$ solve $\mathbb{K}\Lambda = \mathbb{F}$
6: $\widehat{U} \leftarrow \Lambda \cup g_D$
7: **for** $K \in \mathcal{T}_h$
8: reconstruct $\boldsymbol{q}_h^K$, $u_h^K \leftarrow \begin{bmatrix} A & B \\ -B^T & D \end{bmatrix}^{-1} \left(\begin{bmatrix} 0 \\ F \end{bmatrix} + \begin{bmatrix} C \\ E \end{bmatrix} \widehat{u}_h^K \right)$.
9: **end for**

Since exact polynomial tests can be run very cheaply, they are our integration test of choice for the following series of automated tests. For convenience, we reproduce the serial HDG algorithm for reference in Algorithm 7. The tests shown are 2D for simplicity of manufactured solution, but their equivalent 3D test cases should be immediately apparent.

3.2.1 Reconstruction tests

The simplest HDG test case is the element-local reconstruction of the numerical solution from the global flux $\widehat{u}_h$. These tests involve computing $\widehat{u}_h$ using the analytical solution u, and computing $\boldsymbol{q}_h$ and u_h from step 8. This tests the machinery of the element-local equations and the integral operators A, B, C, D, and E without having to form or solve a linear system; reconstruction tests will pass even with a bug in the global index map or assembly of the linear system.

To provide an example of a reconstruction test, we solve the steady-state diffusion

equation

$$\begin{aligned} -\nabla \cdot (\nabla u) &= f \quad \text{in } \Omega, \\ u &= g_D \quad \text{on } \Gamma, \end{aligned} \tag{3.1}$$

on $\Omega = (-1, 1) \times (-1, 1)$, with the source term f and g_D chosen such that we have the following exact solution:

$$u = (x + 1)(x - 1)(y + 1)(y - 1). \tag{3.2}$$

In order to ensure $u \in V_h^p$, we discretize using a single element mesh at polynomial order $p = 4$. The computational mesh and numerical solution are shown in Figure 3-1.

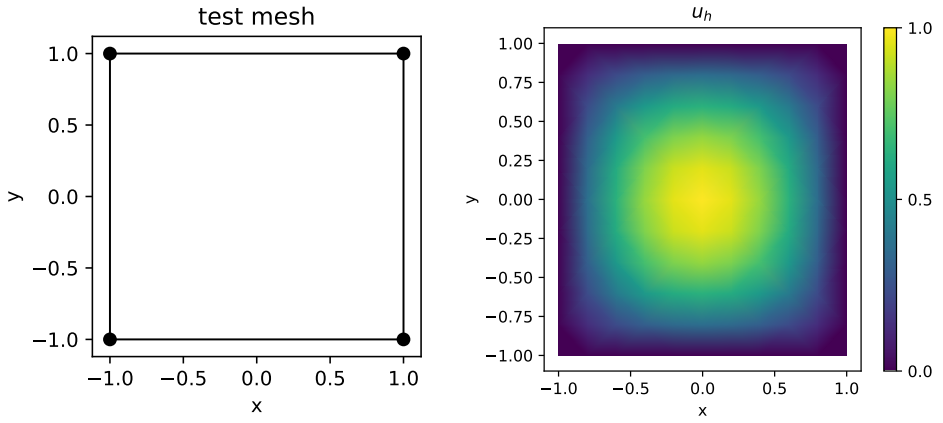


Figure 3-1: Computational mesh (left) and numerical solution u_h (right) for the reconstruction test with manufactured solution u given in equation (3.2).

The boundary conditions are Dirichlet, and homogeneous ($g_D = 0$) analytically. All elemental transformation information is the identity since the element being tested coincides with the master element; therefore, incorrect handling of the isoparametric transformation data J will still allow this test to pass. In some ways, this test constitutes the simplest possible HDG test case.

An analogous case can be constructed for triangular elements by choosing

$$u = -y(x - y)(x + y - 2) \tag{3.3}$$

on $\Omega = \Delta((0, 0), (2, 0), (1, 1))$ where Δ denotes the interior of the simplex defined by the three argument points, and where $u \in V_h^p$ is ensured by choosing polynomial order $p = 3$. However, this test case additionally tests the handling of the isoparametric transformation J, since Ω no longer coincides with the master element.

Analogous test cases in 3D are found by straightforward modification of the analytical solutions u to guarantee $g_D = 0$ on the hexahedral, wedge, and tetrahedral master elements and an appropriately high polynomial order to ensure $u \in V_h^p$.

If the reconstruction tests pass, we can be confident that the elemental contributions $\mathbb{K}^K$ and $\mathbb{F}^K$ are also correct for any test case with homogeneous Dirichlet boundary conditions ($g_D = 0$), since they involve the same operators.

3.2.2 Linear system tests

The next step in the HDG testing hierarchy targets the assembly and solution of the linear system $\mathbb{K}\boldsymbol{\Lambda} = \mathbb{F}$. Since we argued that the passing of the reconstruction tests verifies the element-local contribution matrices $\mathbb{K}^K$ and $\mathbb{F}^K$, these tests mainly serve to detect an error in the local-to-global index map, which determines the assembly of the linear system.

We once again solve equation (3.1) with manufactured solution u given in equation (3.2), the problem described in section 3.2.1. To test the smallest possible linear system, we discretize the problem on a two-element mesh shown in Figure 3-2; that way, there is only one single interior edge.

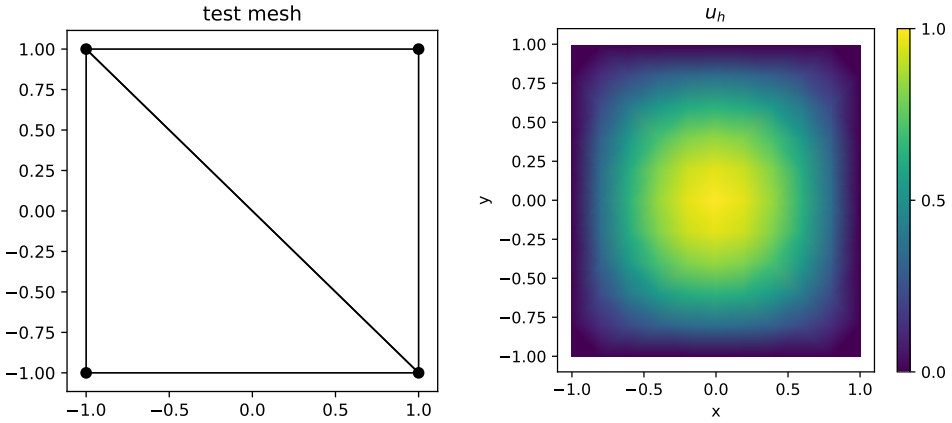


Figure 3-2: Computational mesh (left) and numerical solution u_h (right) for the simple linear system test with manufactured solution u given in equation (3.2).

The next test in the hierarchy is to solve the same problem, but with more elements and solved on a mixed mesh; an example is shown in Figure 3-3. This verifies that the index maps remain correct with multiple elements, and that the handling of different element types is correct. Because the boundary conditions are everywhere Dirichlet and analytically homogeneous, these tests will pass even if the boundary terms are incorrectly implemented.

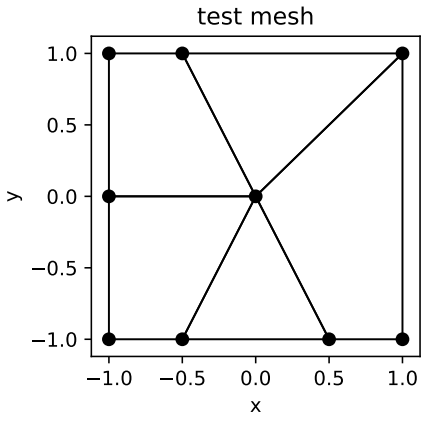


Figure 3-3: Computational mesh for linear system test testing multiple elements on a mixed mesh.

Extension of the test to 3D can be achieved by extruding the 2D meshes to a single-element flat- bottomed 3D mesh, and modifying u to vanish on the 3D boundaries and increasing the polynomial order such that $u \in V_h^p$.

3.2.3 Boundary condition implementation tests

Chapter 1 detailed the boundary condition treatment handling carefully so that all Dirichlet data was applied to the right-hand side, excluding the edges on Γ_D from the approximation space (see equations (1.20) and (1.21)). Inhomogeneous Dirichlet boundary values $g_D \neq 0$ appeared in the F and R operators, which are used to form the elemental contributions $\mathbb{F}^K$ and are assembled into $\mathbb{F}$. To test the implementation of these terms, we solve the same problem

$$\begin{aligned} -\nabla \cdot (\nabla u) &= f \quad \text{in } \Omega, \\ u &= g_D \quad \text{on } \Gamma, \end{aligned} \tag{3.4}$$

on $\Omega = (-1, 1) \times (-1, 1)$, with the source term f and g_D chosen such that we have the following exact solution:

$$u = x + y. \tag{3.5}$$

Here, the manufactured solution u is such that $f = 0$ but $g_D \neq 0$. Any multiple-element computational mesh will do; our test cases use the one shown in Figure 3-2. To ensure $u \in V_h^p$, we choose polynomial order $p = 1$. Since we are confident that the interior edges are being handled correctly from the linear system tests in section 3.2.2, this test isolates the implementation of the inhomogeneous Dirichlet boundary conditions. However, if the Neumann boundary conditions are incorrectly implemented, this test will still pass.

To test the Neumann boundary condition handling (the L operator and its assembly), we extend the same test case, by making the boundary described by $x = 0$ part of Γ_N. To test everything working together in conjunction, we choose the source term f, and the boundary values g_D and g_N to coincide with the more complicated manufactured solution

$$u = x^3 - y^3 + xy, \tag{3.6}$$

solved on the mixed mesh in Figure 3-3. Extension to 3D problems can be accomplished by extruding the mesh as discussed previously.

The basic machinery of the HDG spatial solves can be verified if all these tests pass. Further, since all are exact polynomial tests, they can be executed quickly. Running the entire test hierarchy in serial along with basic variations (non-affine meshes and variable diffusivity κ) for 2D and 3D test cases can be executed in approximately 10 seconds on a desktop machine. Further, the `pytest` library can collect and run all the tests automatically every time a change is committed to the code base, generating a report of which tests passed and failed, as shown below.

```
========================== test session starts ==========================
platform -- Python 3.6.6, pytest-3.7.2, py-1.5.4, pluggy-0.7.1
rootdir: /MSEAS-3DHDG/test/hdgTests, inifile:
collected 19 items

test_run_kernel_test_cases.py ..................                  [100%]

======================== 19 passed in 8.03 seconds ========================
```

3.2.4 Automated convergence tests

The testing hierarchy can be supplemented with automated convergence test cases by choosing a manufactured solution $u \notin V_h^p$. Since convergence tests are more expensive, they should give the maximum amount of code coverage per test. Our convergence test cases test problems on non-affine mixed element meshes with inhomogeneous Dirichlet and Neumann boundary conditions in 2D and 3D. This has the advantage of catching subtler problems, such as incorrect projection of inhomogeneous boundary values $\mathsf{P}g_D$, which would not affect an exact polynomial test, but would pollute the convergence order of an HDG scheme. These tests can be run in the background upon completing the HDG testing hierarchy.

3.3 Summary

Chapters 1 and 2 together detailed implementation of an efficient HDG software kernel, which can take as input boundary conditions $g_D(t)$ and $g_N(t)$ and return the solution variables $\boldsymbol{q}_h$, u_h, and $\widehat{u}_h$ for the model problem given in equation (1.33) at some time t. Such a kernel can be used in conjunction with any suitable time-marching scheme, requiring as parameters only the boundary condition values and time integration coefficients (see, for example, the IMEX coefficients described in Ascher et al. [3]).

In this chapter, our contribution is a verification procedure to test the features of the HDG kernel software in a methodical and automated way. To that end, we developed a framework that can test the functionality of an HDG solve inexpensively and in a hierarchical manner. Doing so provides verification that changes to the codebase have not broken the functionality of the HDG solve and identifies the parts of the software that are responsible for a bug, should a subset of the test cases fail. The testing framework can easily incorporate convergence testing in both space and time to verify the optimal convergence properties of the solver; the convergence testing can be made efficient by *a posteriori* identification of the asymptotic convergence regime. Further, the testing framework can be used for other problems using the HDG framework including advection problems treated explicitly [85] or implicitly [58, 59]. The novelty of our verification approach is that it is specific to the HDG algorithm.

Employing the automated testing practices herein provides software verification and eliminates time-intensive and repetitive running of previously working test cases upon changes to the code, increasing the maintainability and robustness of the numerical software.

Chapter 4

Visualization of Discontinuous Finite Element Data

The convergence results presented in Chapter 1 demonstrate that high-order methods can be much more accurate for the same efficiency as a comparable low-order polynomial solution on a refined grid. However, in the context of computational fluid dynamics or ocean modeling [22, 22, 35, 33], it is nearly always the case that the terminal use of simulation data is interpretation by a scientist or engineer as part of a larger effort to understand or predict physical phenomena. It is therefore crucial to ensure that the richness of the finite element solution is not lost in the visualization of the solution data; otherwise, the effort and computational expense of using a high order solution is wasted.

Visualizing finite element data poses a unique set of challenges. The most general issue is that the polynomial solution obtained by a finite element is represented by nodal coefficients which specify a linear combination of known non-linear basis functions (for polynomial solutions of order $p \geq 2$). Many scientific visualization technologies are aimed at low-order schemes: they transform high-order finite element data to low-order representations, such as visualizing only the nodal coefficients. We will see that the naive approach of visualizing the nodal coefficients without incorporating the information of the basis functions is insufficient and does not, in general, capture the information in the high order solution. This not only adds a so-called "visualization error" to the total error of the solution, but also burdens the scientist or the researcher with the task of determining whether features of the simulation are physical or are simply visualization artifacts [39].

Moreover, an attractive feature of discontinuous Galerkin schemes is the ability to capture steep gradients by admitting discontinuous solutions, but the discontinuous nature of the solution naturally poses visualization challenges, especially in 3D. Most existing visualization frameworks are developed with continuous data in mind [80] and require interpolations that may spoil the integrity of discontinuities between elements, even if a high order solution is preserved on each element interior. The technologies for visualizing such output are simply not mature, and are an active area of development for many state-of-the-art codes [7].

In this chapter, we will justify our claims regarding the necessity of using visualization techniques specifically aimed at high-order finite-element data. We will present 2D and 3D solutions to the challenges addressed above, as well as ways to visualize data unique to HDG finite elements, such as visualizing data on the element edge space and on the global edge space. We will conclude with suggestions and best practices, as well as a brief discussion of

emerging visualization technologies.

4.1 Visualizing high-order polynomial data

A nodal finite element solution is a linear combination of the global basis functions weighted by some coefficients $u_h = \sum_{i=1}^{g_b} c_i \boldsymbol{\phi}_i$, where g_b denote the number of global basis functions. In the context of a nodal discontinuous Galerkin or hybridizable discontinuous Galerkin formulation, the global basis functions are the union of the nodal shape functions on each element $K \in \mathcal{T}_h$ and only have support on the element on which they reside; that is,

$$u_h = \bigcup_{K \in \mathcal{T}_h} \sum_{i}^{n_b} c_i [\boldsymbol{\phi}_i]_K, \tag{4.1}$$

where n_b denotes the number of nodal shape functions on each element. Since a discontinuous Galerkin finite element solution can be thought of as a continuous polynomial solution on each element, we begin by addressing how to visualize a continuous solution on a single element. That way, if a plotting software is able to render the solution satisfactorily on a single element, the procedure can be repeated for every element in the mesh, and the discontinuous solution will be well-represented visually. We make the assumption that a plotting agent will render a linear interpolation between the data points explicitly stored.[1]

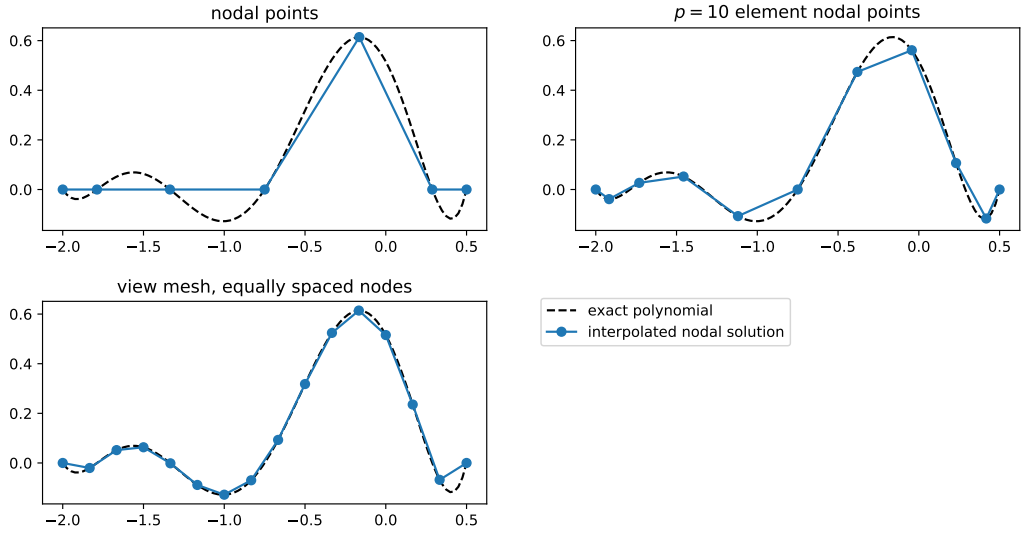


Figure 4-1: Order $p = 6$ polynomial represented by linear interpolations on different view meshes: at the nodal points (top, left), at the nodal points of a higher order ($p = 10$) element (top, right), and on a view mesh of 16 evenly spaced points (bottom, left).

Consider the 1D example of a polynomial solution at order $p = 6$ over an element $K = [-2, 0.5]$. Figure 4-1 shows three different visualizations of the polynomial solution over the element. The plot of the solution at the nodal points of K is the linear interpolation of the nodal values and is insufficient to represent the structure of the solution: plotting the nodal coefficients shows the solution to be zero everywhere except at a single node. This

[1] While many popular plotting and visualization softwares support quadratic and cubic interpolation, these implementations tend to be very expensive, especially in 2D and 3D.

is an example of transforming high-order finite element data to a low-order representation alluded to above, and is to be avoided. A better approach is to evaluate the solution at additional points on K.

At this point, we introduce some terminology in the interest of clarity. We will refer to the linear interpolation of the nodal data alone as "nodal interpolation." Any set of visualization points different than the set of only the nodal points will be referred to as a "view mesh." The two alternatives to the nodal interpolation shown in Figure 4-1 are both examples of view meshes (in this case, the view mesh contains only one element). In the case of a view mesh which contains the nodal points of a higher order element, such as the $p = 10$ view mesh, we will refer to the "viewing/plotting order" of the visualization. We will denote a view order as p_{view} for the sake of avoiding ambiguity between the view order and polynomial order of the numerical solution. It is often convenient to use these particular types of view meshes, especially when the data structures associated with higher order elements are accessible. Refinement of a view mesh for the purposes of visualization will be referred to as "visual refinement" whereas refinement of the computational mesh as used in computing the numerical solution will be referred to as "numeric refinement." If the features of a finite element solution are insufficiently resolved due to not having a fine enough view mesh, as in the nodal interpolation visualization in Figure 4-1, we refer to the solution as "visually under-resolved."

We remark that the isoparametric representation of the mapping between the master element and each physical element allows a straightforward implementation of evaluating the finite element solution on a view mesh. Because of (4.1), we need only evaluate the shape function Vandermonde matrix $\widehat{\boldsymbol{\theta}}_{ij} = \boldsymbol{\phi}_i(\boldsymbol{\xi}_j)$ as defined in Chapter 1 at the locations $\boldsymbol{\xi}_j$ on the master element corresponding to the view mesh. However, a plotting agent must know the locations of all view mesh points. The consequence of expressing the map from master element to physical element as

$$\boldsymbol{x}(\boldsymbol{\xi}) = \sum_{j=1}^{n_b} \boldsymbol{x}_i^K \boldsymbol{\psi}_i(\boldsymbol{\xi}),$$

where $\boldsymbol{x}^K$ are the nodal locations on the physical element K, is that we can map $\boldsymbol{\xi}_j$ to their physical space locations $\boldsymbol{v}_j^K$ by evaluating the matrix-vector product $\widehat{\boldsymbol{\theta}}\boldsymbol{x}^K$, or all at once with the product $\widehat{\boldsymbol{\theta}}X$, where the columns of X are $\boldsymbol{x}_K$.

Naturally, there is a trade-off between the cost of evaluating (and potentially storing) the solution on any type of view mesh and sufficiently representing the solution visually. In general, the objective is to use a fine enough view mesh such that the solution is not visually under-resolved, but not finer, lest computation, memory, and disk space be needlessly wasted.

4.1.1 Quantitatively measuring visual resolution

While finding a good regime of visual refinement is more of an art than a science, here we propose a heuristic, quantitative approach to ensuring a visually-resolved solution. The idea is simple; find a "stationary point" in the visualization up to some tolerance. We can do so by computing the error in the L^2-norm between the difference of the solution as computed on successive view meshes. This is depicted in Figure 4-2 for the 1D example introduced in Figure 4-1. Consider two view meshes, described by the point sets $\{\boldsymbol{x}_1\}$, $\{\boldsymbol{x}_2\}$. We wish to

compute

$$\|u_h(\boldsymbol{x}_1) - \mathcal{I}\left(u_h(\boldsymbol{x}_2)\right)\|_{L^2(\Omega)} = \sqrt{\int_\Omega |u_h(\boldsymbol{x}_1) - \mathcal{I}\left(u_h(\boldsymbol{x}_2)\right)|^2 \, d\Omega}\,, \tag{4.2}$$

where $\mathcal{I}$ is the linear interpolation operator. When this error quantity is below a certain tolerance, the visual appearance of the plotted solution can be considered stationary, and any additional refinement is visually unnecessary.

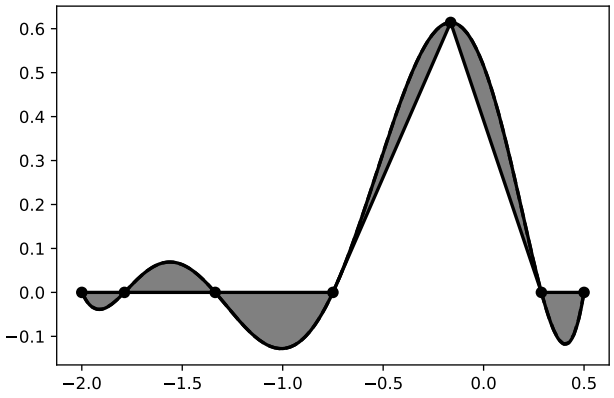
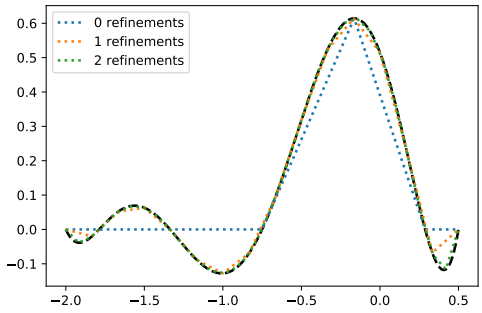


Figure 4-2: Measuring volume between solutions on different view meshes.

To provide a concrete example, we apply this approach to the same 1D polynomial solution, using a set of successive view meshes refined by bisection. We compute the integrals using the trapezoidal rule (note the asymptotic second-order convergence), and scale the integral by the volume of K — we will later make an argument that all integrations can be made over the master element directly, since we are only concerned with the differences between visual representations. By inspecting the error in conjunction with the plots of the refinement, we see that an error threshold of approximately 5e-02 is more than sufficient for visual resolution. If we compute all integrals over the master element, this threshold value generalizes well to other 1D problems.



(a)

h	$\|u_h(\boldsymbol{x}_{i+1}) - \mathcal{I}\left(u_h(\boldsymbol{x}_i)\right)\|$	Order
2.12e-01	1.14e-01	-
1.67e-01	4.38e-02	3.95
8.06e-02	1.16e-02	1.83
3.97e-02	2.85e-03	1.98
1.97e-02	6.99e-04	2.01
9.80e-03	1.72e-04	2.01

(b)

Figure 4-3: (a) Visualization of the polynomial solution on view meshes $\boldsymbol{x}_i$ successively refined by bisection. (b) Convergence history (to zero) of the "visualization error" (4.2).

While any numerical quadrature will suffice for the purposes of computing (4.2), we revisit the argument that use of view meshes corresponding to higher-order finite elements

are particularly useful. If the data structures for higher-order master elements are available for view orders $p_{view} = \{p_1, p_2\}$, the mass matrices M and nodal shape function matrices $\widehat{\boldsymbol{\theta}}_{ij} = \boldsymbol{\phi}_i(\boldsymbol{\xi}_j)$ as defined in Chapter 1 can be used to compute (4.2) in a quadrature-free manner:

$$\|u_h(\boldsymbol{x}_1) - \mathcal{I}\left(u_h(\boldsymbol{x}_2)\right)\|_{L^2(\widehat{K})} \approx M_{p_1}\left(\widehat{\boldsymbol{\theta}}_{p_1}\boldsymbol{c}_K - \mathcal{I}\left(\widehat{\boldsymbol{\theta}}_{p_2}\boldsymbol{c}_K\right)\right), \tag{4.3}$$

where $\boldsymbol{c}_K$ are the nodal coefficients on element K. This approach is more efficient because all master element data can be computed once for different candidate view meshes and re-used, potentially at every time step, since the view mesh required for avoiding a visually under-resolved solution are dependent on problem dynamics. Additionally, the numerical integrals are more accurate and converge faster than if computed with a generic numerical quadrature (such as trapezoidal rule); for justification, see the quadrature-free convergence studies in Chapter 1. Using view orders rather than bisected or equally spaced view-meshes, it is straightforward and efficient to implement automated visual refinement by only specifying a tolerance.

Heuristic improvements to this procedure are easy to conceive for HDG solutions, in particular. All integrations can be computed on the master element rather than over the physical domain, since we are concerned only with the difference between view mesh solutions. Moreover, since the weak gradient $\boldsymbol{q}_h$ is solved for and used, the element with the largest gradient magnitude is a good candidate for a test element on which to compute an acceptable view-mesh, without requiring the integrations on the other elements.

4.1.2 Visual resolution in higher dimensions

In higher dimensions, the ideas are the same, but we need to consider how the patches or volumes are shaded. The principle is the same as in 1D; the finite element solution is evaluated on a view mesh. However, in 2D and 3D, that view mesh is divided into simplices with a Delaunay triangulation, and individual simplex elements are shaded according to the plotting agent. Flat shading renders each simplex as a single color; shading is constant over the simplex. Gouraud shading computes a color at each vertex and interpolates linearly between the colors across the simplex. In 2D and 3D, the effects of visual under-resolution are typically more pronounced than in 1D cases.

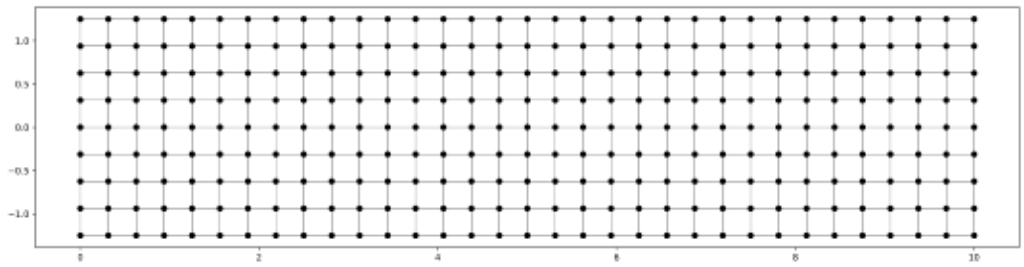
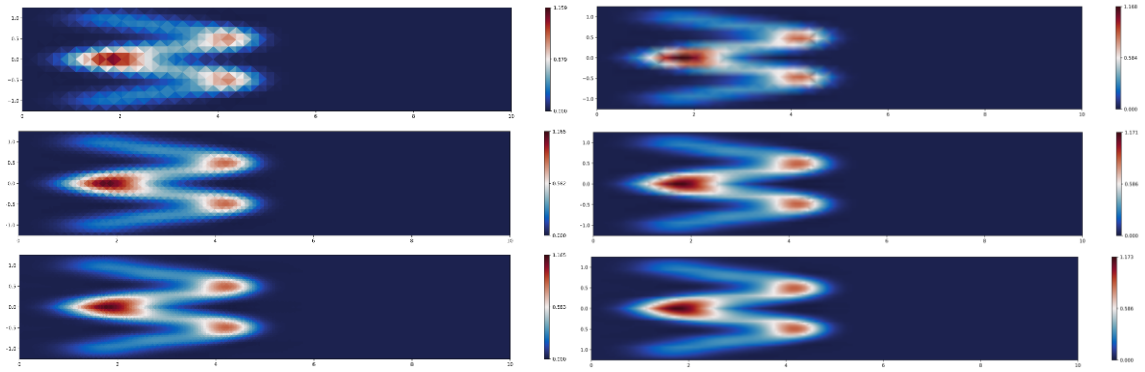


Figure 4-4: Computational mesh used in domain discretization.

Figure 4-5 shows a tracer field visualized on the structured mesh of quadrilateral elements shown in Figure 4-4 shows a mesh of quadrilateral elements, but on different view meshes. Although all subplots correspond to the same numerical solution, the nodal interpolations are severely visually under-resolved. By contrast, the intermediate and fine view

meshes ($p_{view} = 4$ and $p_{view} = 6$, respectively) are not visually under-resolved. Note that the differences between shading types become less noticeable once the solution is visually resolved.



(a) flat shading (b) Gouraud shading

Figure 4-5: All plots show numerical solution order $p = 2$ on the computational mesh in Figure 4-4, (top) viewed with nodal interpolation, (middle) visualized at $p_{view} = 4$, (bottom) visualized at $p_{view} = 6$.

The goal of avoiding visually under-resolved solutions is not an inconsequential one. Visualizing a numerical solution can generally be considered as post-processing. Any visualization computations can be done completely in parallel and without any data coupling; it will not be a computational bottleneck. It is important to avoid numerically refining a high-order solution until it is also visually refined — doing so is not only costly in terms of the numerical bottlenecks, but also completely defeats the purpose of using a high-order solution. Refining the actual computational grid until a nodal interpolation visualization appears resolved is essentially refining to the point that the high order solution on each element is approximately linear.

4.2 Visualizing discontinuous data

In section 4.1, we discussed strategies for visualization of a high order polynomial solution. Now we turn our attention to strategies for visualizing discontinuous data. The central idea is encapsulated in (4.1); we must simply have the plotting agent render each element individually. In 2D, this corresponds to "patch plotting," where a continuous solution is drawn on each element independently. In 3D, this generalizes to volumetrically rendering the solution on each element separately. Within each element, any of the aforementioned techniques for rendering a high-order polynomial solution can be employed. Until this point, we have deferred discussion of the particular softwares used as plotting and visualization agents, but in this section we will address the software choices we make.

For 2D patch plots, the open source `matplotlib` library [37] is often sufficient for visualizing finite element data. However, for "height plots", which display a 2D scalar value as an embedded surface in $\mathbb{R}^3$ where the height of the surface corresponds to the 2D scalar value, as well as for all 3D visualizations, we use the Visualization Toolkit (VTK) library and open-source software Paraview for visualization [77, 6] . We defer the details of this visualization pipeline until section 4.3, where it is more relevant.

4.2.1 Patch plotting and height plotting

For 2D numerical solutions, patch plotting is the simplest choice for visualization of a discontinuous finite element solution. If the solution is sufficiently smooth and visually resolved, patch plotting can adequately capture the features of the numerical solution.

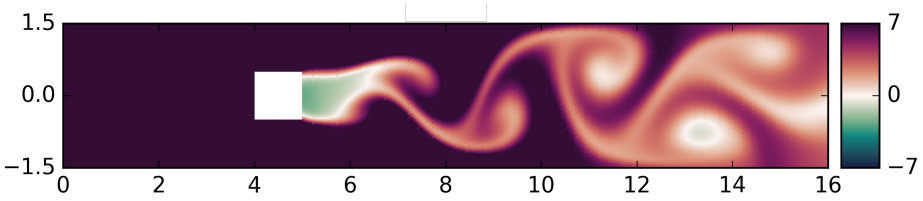


Figure 4-6: `matplotlib` rendered patch plot visualization of a passive tracer field in a flow field around a square cylinder

Figure 4-6 shows a `matplotlib` passive tracer field in a flow around a square cylinder [32], in which vortex shedding can be observed. The field is visually well-resolved and the solution itself is smooth. In such cases, patch plotting is often sufficient to visualize the solution.

However, patch plotting can be insufficient to visualize solutions with discontinuities or with steep gradients; in such cases, height plots are often useful to visualize both. In Figure 4-7, a patch plot and a height plot are used to visualize the same ($p = 0$) piecewise constant discontinuous Galerkin approximation to the function $u = 2 - x^2 - y^2$ on the domain $\Omega = [-1, 1] \times [-1, 1]$.

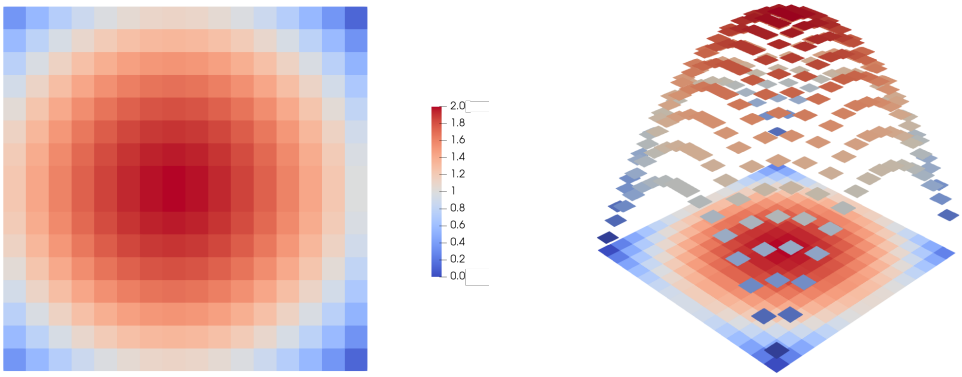


Figure 4-7: Paraview rendering of both a patch plot (left) and a height plot (right) of the $p = 0$ discontinuous Galerkin approximation to the function $u = 2 - x^2 - y^2$, colors scaled from $[0, 2]$.

Note that by viewing only the 2D patch plot, it's unclear if the plot is visually under-resolved or if the solution is discontinuous; a height plot immediately reveals the latter. Furthermore, the relative magnitudes of the discontinuities are not visually apparent from the patch plot, but are immediately clear from the height plot.

Figure 4-8 depicts the numerical solution to the time-dependent heat equation with a `matplotlib`–rendered height plot. An HDG solver using linear ($p = 1$) quadrilateral elements was used to compute the numerical solution. The colorbar and the height of each element both indicate the value of the solution, but the height plot emphasizes the discontinuities between the elements.

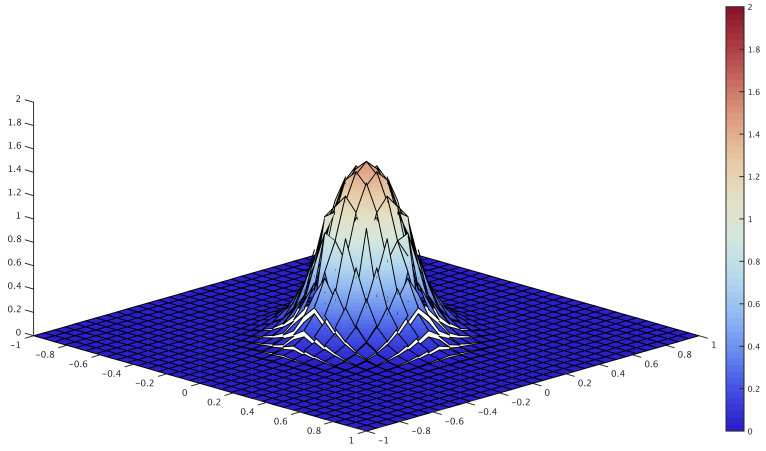


Figure 4-8: Visualizing discontinuous solutions with a height plot with `matplotlib`.

The same limitations of visual resolution and techniques for rendering high-order polynomial solutions can be applied to patch plots and height plots. As an example unifying all of the techniques we have described thus far, we consider the discontinuous function

$$u(x, y) = \sin(2\pi x)\sin(2\pi y) + \mathbf{1}_I(x, y) - \mathbf{1}_{III}(x, y), \tag{4.4}$$

on $[-1,\, 1]\times[-1,\, 1]$ where $\mathbf{1}_I$ and $\mathbf{1}_{III}$ are the indicator functions for quadrant I and quadrant III of $\mathbb{R}^2$, respectively. We consider the piecewise discontinuous Galerkin approximation of u, $u_h = \mathsf{P}u$, at polynomial order $p = 4$ (where P denotes the L^2 projection into the space W_h^p as defined in Chapter 1) on the mesh shown in Figure 4-9. Note that the mesh contains one finite element per quadrant.

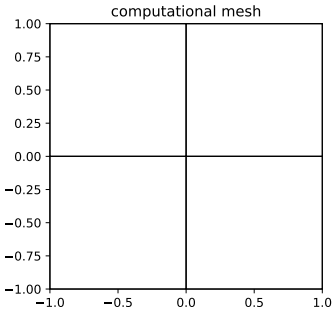
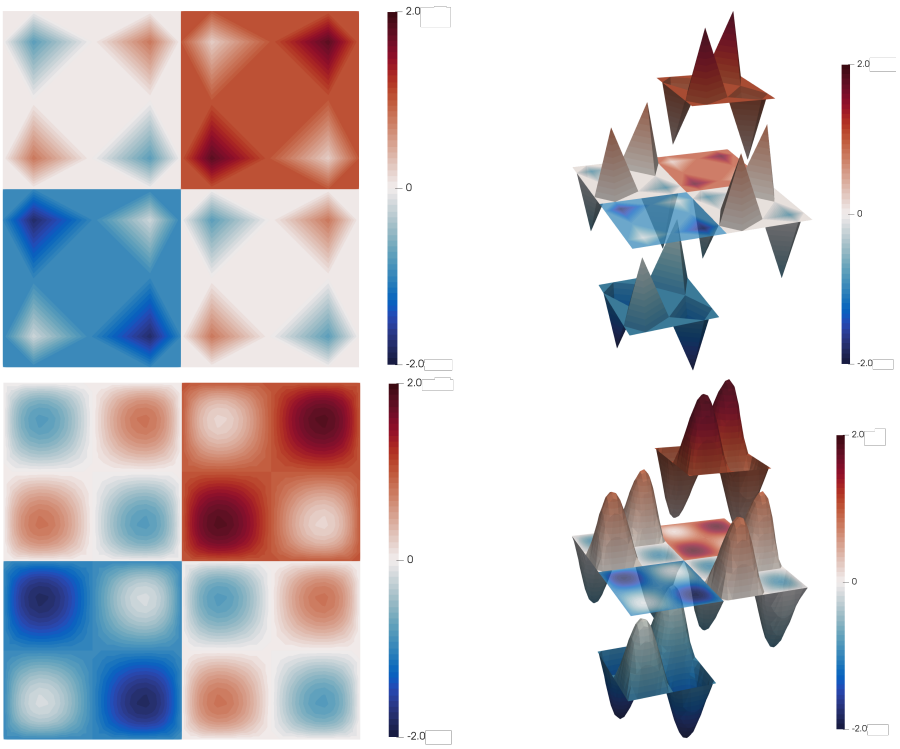


Figure 4-9: Computational mesh for the discontinuous Galerkin projection $\mathsf{P}u$ of $u(x, y)$ defined in (4.4).

Figure 4-10 shows patch plot (4-10a) and height plot (4-10b) visualizations of the discontinuous Galerkin projection $\mathsf{P}u$ of $u(x, y)$. Both the patch plot and the height plot of the nodal interpolation appear undesirably visually under-resolved and make the solution appear under-resolved and sharp, but the discontinuities are well communicated. Applying the procedure described in section 4.1.1, we find that a view mesh at order $p_{view} = 20$ yields a stationary value of the visualization, which can also be seen in Figure 4-10. Indeed, the



(a) Patch plot (b) Height plot

Figure 4-10: The nodal interpolation (top) and visualization on a view mesh at order $p_{view} = 20$ (bottom) of the discontinuous Galerkin projection $\mathsf{P}u$ of $u(x, y)$ defined in (4.4), rendered in Paraview as patch plots and height plots.

features of the polynomial solution on the view mesh are visually well resolved.

We emphasize that, in the context of interpreting a numerical solution, patch plots and height plots simply provide different perspectives. A complete visualization pipeline should involve viewing both types of plots. Figure 4-11 depicts a Paraview rendering of the tracer field $\rho' = (\rho - \rho_0)/(\rho_{max} - \rho_{min})$, a nondimensionalized density perturbation from the mean, in a 2D lock exchange simulation, both as a patch plot and as a height plot. The patch plot and height plots give different viewpoints of the dynamics involved. The patch plot communicates the symmetry and relative locations of the Kelvin–Helmholtz instabilities at the interface of the two fluids of differing densities. The height plot visually communicates the steep gradients around the instabilities and the smoothness of the solution — the latter is an indication to the researcher that perhaps the solution is numerically over-resolved and that the computational mesh can be coarsened.

4.3 3D Visualization

Choices for 3D CFD visualization software are limited compared to 2D plotting software, and the high-order and discontinuous nature of the finite element solutions we have addressed thus far requires additional flexibility. In this section, we describe the software pipeline used to generate high-quality 3D visualizations, and discuss the data structures and approaches necessary to render HDG-specific data.

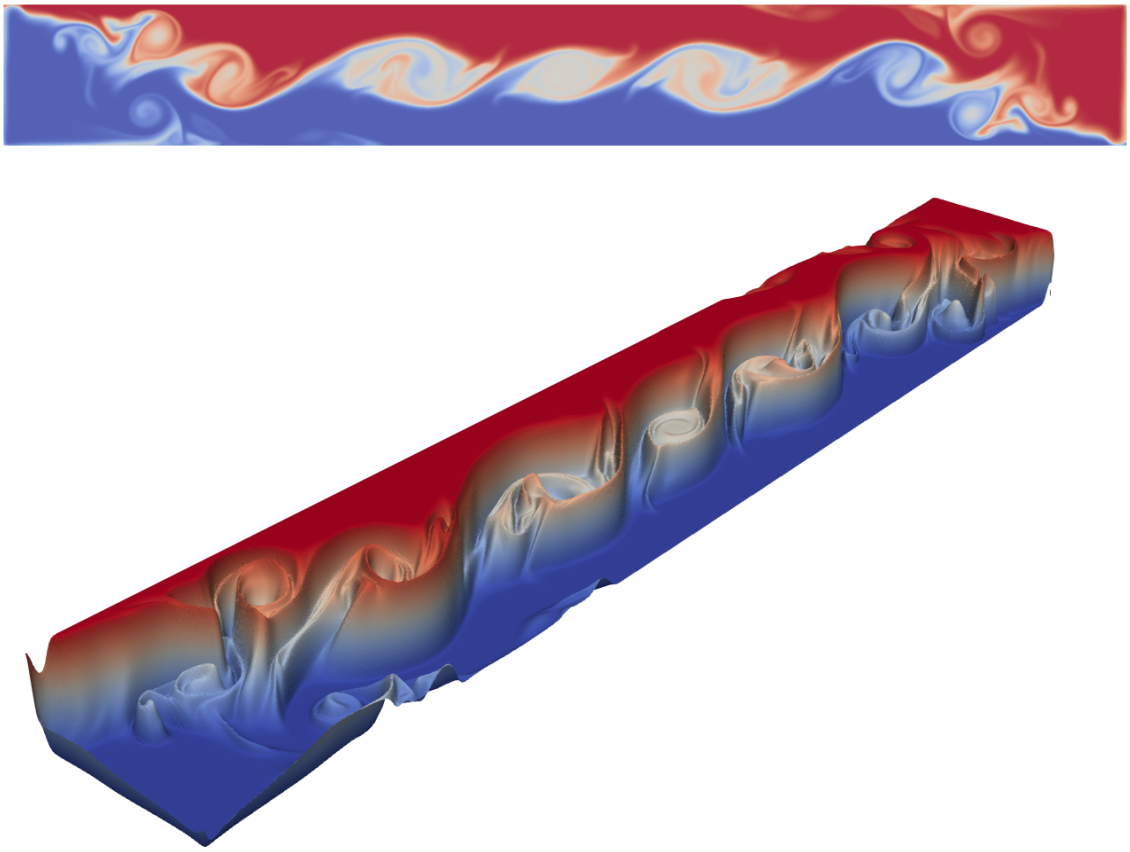


Figure 4-11: Paraview rendering of a 2D patch plot (top) and height plot visualization (bottom) of nondimensionalized fluid density perturbation in a numerical simulation of a 2D lock exchange problem. Red denotes "lighter" density, and blue "heavier" density.

4.3.1 Software pipeline

Discontinuous finite element data requires rendering the polynomial data on each element separately; to do so, we make use of the open-source Visual Toolkit (VTK) library. VTK consists of a mature object-oriented `C++` library and interfaces that allow users to build visualizations by combining various objects together [77]. VTK can be thought of as providing the building blocks and output formats to create custom visualizations. We will discuss the VTK representations of relevant finite element data and the corresponding outputs.

We use a subset of the VTK data model for the purposes of our visualizations. All sets comprised of polygonal data (lines, planes, triangles, point clouds) are represented and output as a `vtkPolyData` object[2]. All polygonal data is assumed to be embedded in $\mathbb{R}^3$. All sets of unstructured volumetric data are represented and output as a `vtkUnstructuredGrid` object[3], which allows for irregular topology and geometries. VTK supports tetrahedral, hexahedral, and wedge cell types, allowing for the finite elements associated with 2D mixed

[2] `vtkPolyData` is an instance of the more general `vtkDataSet`, and its output format is a `.vtp` VTK "PolyData" file. Its parallel equivalent is the VTK "PPolyData" (`.pvtp`) data file.

[3] `vtkUnstructuredGrid` consists of arbitrary combinations of any possible cell type, and its output format is a `.vtu` VTK "UnstructuredGrid" data file. Its parallel equivalent is the VTK "PUnstructuredGrid" (`.pvtu`) data file.

meshes and 3D extruded meshes, which are those relevant to our MSEAS-3DHDG code [83].

Scalar, vector, and tensor-valued data can be added to the different VTK objects. The 3D generalization of the techniques for plotting high-order polynomials over a view mesh in the volume of each element is accomplished by performing a Delaunay triangulation of the view mesh points in each element and representing each simplex as a tetrahedral cell.[4]

Apart from the VTK data model, the VTK library provides functionality for the rendering of all objects and data therein. However, using the VTK library directly to render data visualizations is unnecessary, as the rendering and visualization of scientific VTK data is exactly the purpose of Paraview. Paraview is an open-source software framework that is built on VTK and provides functionality for data analysis and visualization of VTK data; Paraview additionally extends VTK and provides an application/GUI framework for interactive visualization, a Python scripting language for programmatic generation of visualizations, a client-server architecture to facilitate remote visualization of datasets, and supports data parallelism on shared-memory or distributed-memory multicomputer platforms [6]. Paraview is capable of interpreting and rendering any VTK datafile, allowing user configuration of visualization settings. We have now specified a complete visualization pipeline.

4.3.2 Mesh visualization

Visualizing either a 2D or 3D mesh is a matter of writing out each element in the mesh as a VTK cell, as described in section 4.3.1. Figure 4-12 shows the visualization of a simple, extruded mixed mesh.

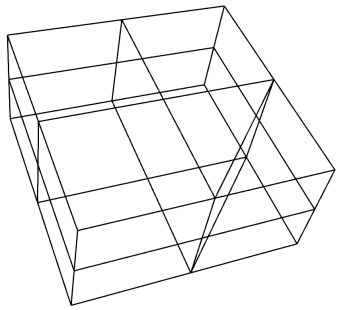


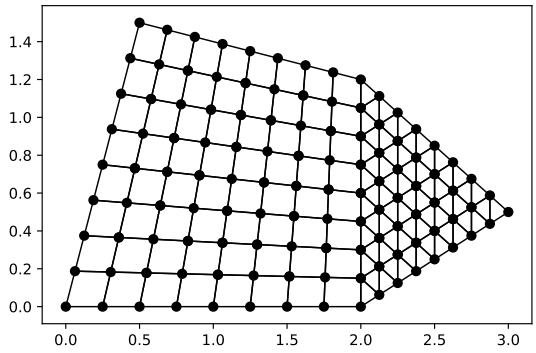
Figure 4-12: Simple extruded mixed mesh.

Figures 4-13a and 4-13b depict top-down and volumetric visualizations, respectively, of an extruded mixed mesh with non-trivial bathymetry.

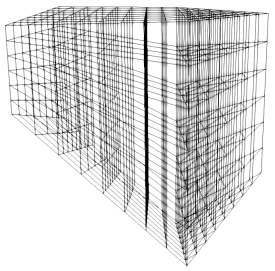
4.3.3 Volumetric visualization

There are several choices for visualization of volumetric 3D data. We give examples of semi-transparent volumetric visualization, cross-section slices, and contour surfaces — a surface

[4] A feature recently implemented in VTK is support for Lagrange cells, which handles the evaluation of the high order polynomial at a view mesh defined by bisection over each element as a rendering feature. However, the feature is still under development and only supports polynomial orders up to 10, which for the time being necessitates writing VTK data onto a view mesh.



(a) Bathymetric, extruded mixed mesh, top-down view.



(b) Bathymetric, extruded mixed mesh with 6 layers.

Figure 4-13

on which the function has a constant value.

Consider the example of a 3D lock exchange with free-slip boundary conditions on the sides of the domain. We will examine different visualizations of the nondimensionalized fluid density perturbation in the numerical simulation. We use a grid with $Ny = 12$, $Nx = 6$, $Nz = 3$ hexahedral elements at order $p = 5$. Figure 4-14 shows a semi-transparent volumetric rendering of the density perturbation with a contour surface depicting the isopycnal surface $\delta\rho' = -0.3$. Figure 4-15 shows the same, but visualized with slices along the vertical and horizontal center-planes (axially) and with the same isocontour of the density perturbation. Both visualizations are rendered on a view mesh of order $p_{view} = 20$.

The volumetric rendering gives a sense of the scalar density perturbation field everywhere in the domain, but would be a poor choice for viewing the visually overlapping dynamics. The slice depiction, on the other hand, gives an incomplete description of the density perturbation over the entire domain, but each slice does not suffer from the problem of rendering visually overlapping dynamics. As in the case of 2D visualization, both methods should be used in conjunction to form a complete visualization of the simulation.

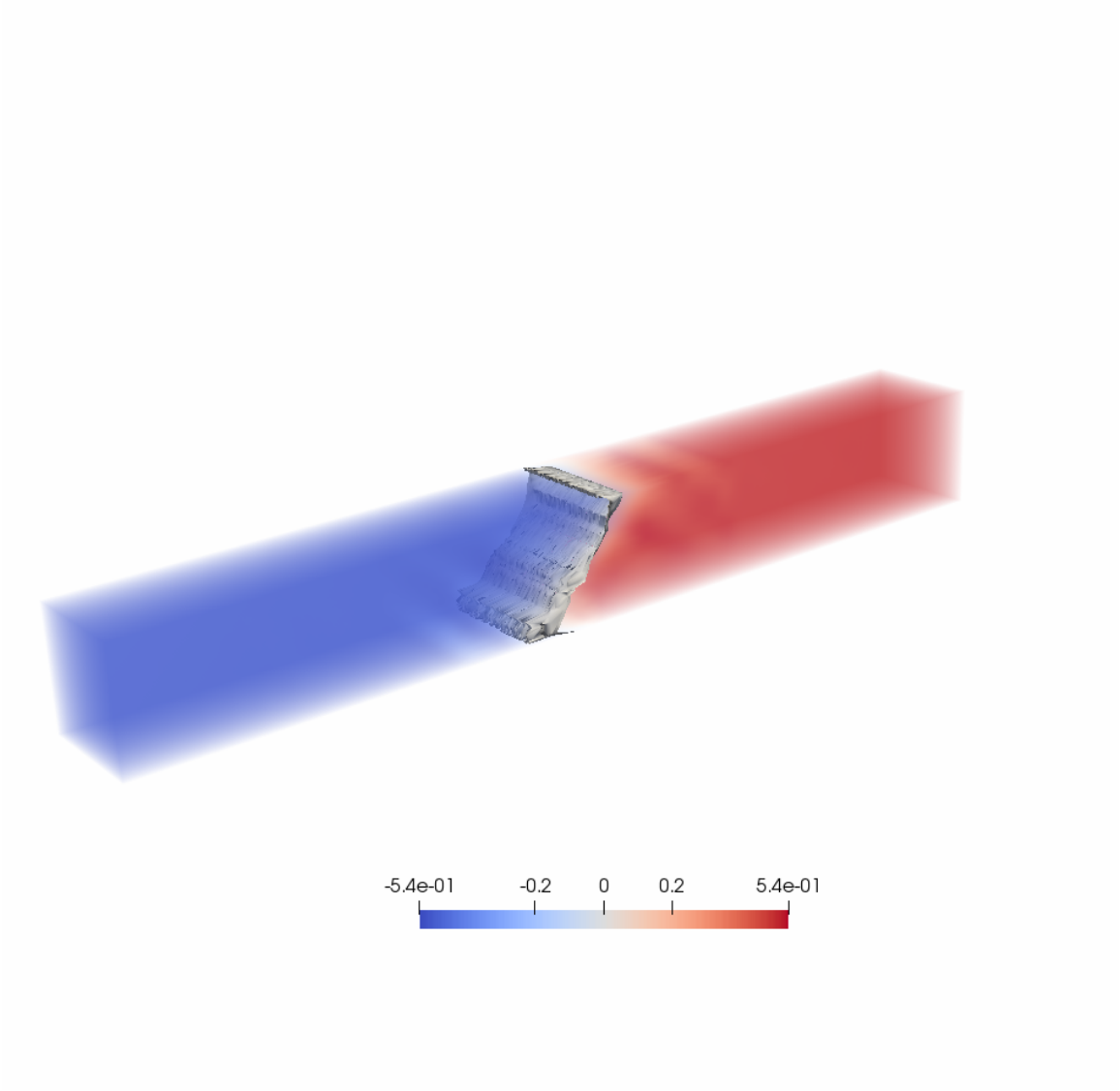


Figure 4-14: 3D Lock exchange volume / contour visualization of normalized density perturbation.

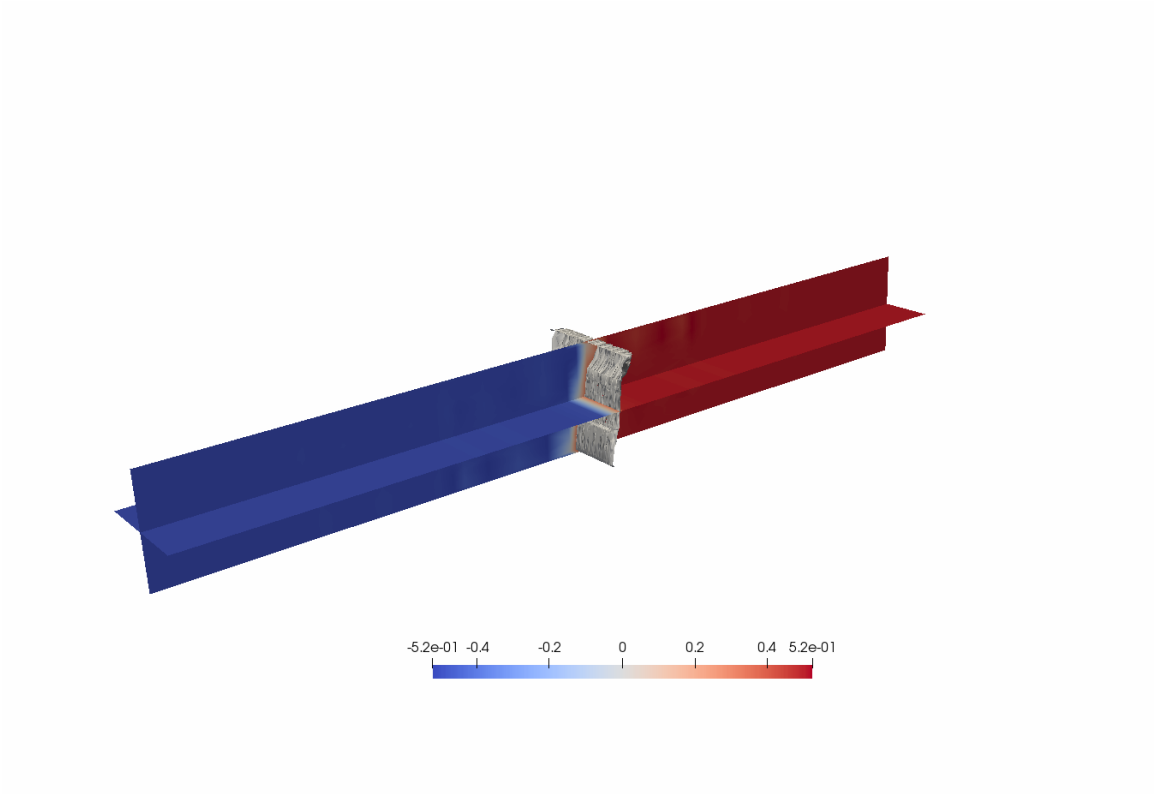


Figure 4-15: 3D Lock exchange slice / contour visualization of normalized density perturbation.

4.4 Summary

In this chapter, we detailed our approach to visualization of discontinuous finite element simulation data. We described generalized approaches to visualize high-order polynomial data, and demonstrated the necessity of avoiding visually under-resolved plots. We showed how to use the isoparametric transformation in Chapter 1 to evaluate polynomial solution on a view mesh — this is a common approach in the finite element community. However, our approach can evaluate the solution on a view mesh by storing only the nodal basis function data on a higher order master element, which differs from other implementations that compute the additional solution points by bisection [68] or interpolate the solution onto a finer mesh which must be stored [44, 54]. We presented novel criteria for quantitatively measuring "visual resolution" and avoiding visual under-resolved solutions automatically; although usually overkill, these algorithms ensure that the user of a finite element codebase does not interpret visually under-resolved solutions as numerically under-resolved.

We described strategies for representing discontinuous data in 2D, and commented on the strengths and weaknesses of patch plots and height plots as a representation of the underlying discontinuous finite element solution. We described our software pipeline for rendering 3D data, inspired by similar cell-based high-order visualization methods [76], and presented examples and techniques for rendering 3D meshes, as well as 3D volumetric data.

Chapter 5

Conclusions and Future Work

Although the parametrization of the model problem into a global equation system for the approximate trace $\widehat{u}_h$ and the introduction of the HDG edge space adds complexity to the standard DG finite element approach, the weak formulation is straightforward. However, there is a leap between the mathematical formulation of HDG methods and the actual implementation of the algorithms. This is particularly true given the rich set of implementation choices which ultimately affect the efficiency, robustness, and flexibility of the resultant schemes.

The first contribution of this work has been to identify a flexible implementation that allows explicit handling of Dirichlet boundary conditions to preserve symmetry of the nonzero entries of the global linear system, as well as a mapping procedure for modular representation of curvilinear meshes that allows for use of quadrature-based and quadrature-free schemes.

Through numerical experiments, we characterized the errors associated with handling of non-affine mixed meshes and demonstrated that the optimal convergence rates are attained with quadrature-free discretization on non-affine and curvilinear meshes, despite claims in the literature [5, 69] that spatially constant Jacobian matrices are necessary for a quadrature-free approach. The numerical test cases both provided verification and demonstrated the accuracy and robustness of the quadrature-free methods.

The second contribution of this work provided an automated and systematic approach to verification of the HDG algorithm. By judicious choice of test case, the testing procedure can verify all portions of the HDG algorithm in seconds, providing useful diagnostic information upon failure of a test. The hierarchical approach to testing lends additional credence to the implementation.

Having demonstrated the viability of the quadrature-free approach, our last contribution uses the form of the discrete integral operators to derive a novel and efficient matrix-free algorithm for solution of the global HDG linear system for the approximate trace λ_h. The approach employs a Schur complement approach exploiting the mathematical structure of the discretized element-local equations. All implementations herein are independent of dimension, and remarks and guidelines are provided for effective visualization of high-order, discontinuous finite element solutions in 2D and 3D.

All contributions contained in this thesis relate to the linear model problem introduced in Chapter 1. The justification for the scope of this consideration is that the HDG software kernels developed in this work form the basis for — and are immediately applicable to — much more complicated problems: the linear and nonlinear convection diffusion equations

[58, 59], projection method [85] and fully-implicit solution [61] of the incompressible Navier–Stokes equations, and the ocean equations [83, 86, 28]. In this sense, scalable, flexible, and efficient implementation of the HDG kernels is extensible and advantageous.

In the future, the present work will complement the suite of multidisciplinary simulation, estimation, and assimilation systems (MSEAS) methods and software [55], used for fundamental research and for realistic simulations and predictions in varied regions of the world's ocean [50, 63, 34, 35, 66, 33, 46, 48]. Applications include monitoring [47], real-time acoustic predictions and data assimilation [45, 89, 43, 24], as well as ecosystem predictions and environmental management [10, 20]. Namely, the research in this work will be applied the MSEAS-3DHDG non-hydrostatic 3D Navier-Stokes and Boussinesq code, developed pursuant to research in high-order regional ocean modeling [84, 83, 85, 86]. These models can be employed for targeted non-hydrostatic or biogeochemical process-studies.

Preliminary work has been done to extend HDG methods to a distributed computing environment [73, 74, 28, 42, 27] , and a logical step for future work involves adapting the matrix-free approaches and implementations in this work to a distributed multicore or GPU environment. The fast application of the sparse HDG matrix-vector product in particular may be amenable to GPU acceleration, perhaps in conjunction with an HDG-specific graph-coloring approach similar to that discussed in Roca [74] to provide further vectorization. Efficient, scalable approaches to high-order schemes could then be employed for multi-scale ocean and fluid modeling [21].

Bibliography

[1] Douglas N Arnold. An interior penalty finite element method with discontinuous elements. *SIAM journal on numerical analysis*, 19(4):742–760, 1982.

[2] Douglas N Arnold, Franco Brezzi, Bernardo Cockburn, and L Donatella Marini. Unified analysis of discontinuous Galerkin methods for elliptic problems. *SIAM journal on numerical analysis*, 39(5):1749–1779, 2002.

[3] Uri M Ascher, Steven J Ruuth, and Raymond J Spiteri. Implicit-explicit Runge-Kutta methods for time-dependent partial differential equations. *Applied Numerical Mathematics*, 25(2):151–167, 1997.

[4] H Atkins and H Atkins. Continued development of the discontinuous Galerkin method for computational aeroacoustic applications. In *3rd AIAA/CEAS Aeroacoustics Conference*, page 1581, 1997.

[5] Harold L Atkins and Chi-Wang Shu. Quadrature-free implementation of discontinuous Galerkin method for hyperbolic equations. *AIAA Journal*, 36(5):775–782, 1998.

[6] Utkarsh Ayachit. *The Paraview guide: a parallel visualization application*. Kitware, Inc., 2015.

[7] W. Bangerth, R. Hartmann, and G. Kanschat. deal.II – A general purpose object oriented finite element library. *ACM Trans. Math. Softw.*, 33(4):24/1–24/27, 2007.

[8] Francesco Bassi and Stefano Rebay. A high-order accurate discontinuous finite element method for the numerical solution of the compressible Navier–Stokes equations. *Journal of computational physics*, 131(2):267–279, 1997.

[9] Peter Bastian, Felix Heimann, and Sven Marnach. Generic implementation of finite element methods in the distributed and unified numerics environment (DUNE). *Kybernetika*, 46(2):294–315, 2010.

[10] Ş. T. Beşiktepe, P. F. J. Lermusiaux, and A. R. Robinson. Coupled physical and biogeochemical data-driven simulations of Massachusetts Bay in late summer: real-time and post-cruise data assimilation. *Journal of Marine Systems*, 40–41:171–212, 2003.

[11] F Brezzi, D Boffi, L Demkowicz, RG Durán, RS Falk, and M Fortin. *Mixed finite elements, compatibility conditions, and applications*. Springer, 2008.

[12] Bernardo Cockburn. Static condensation, hybridization, and the devising of the HDG methods. In *Building bridges: connections and challenges in modern approaches to numerical partial differential equations*, pages 129–177. Springer, 2016.

[13] Bernardo Cockburn, Bo Dong, and Johnny Guzman. A superconvergent LDG-hybridizable Galerkin method for second-order elliptic problems. *Mathematics of Computation*, 77(264):1887–1916, 2008.

[14] Bernardo Cockburn, Jayadeep Gopalakrishnan, and Raytcho Lazarov. Unified hybridization of discontinuous Galerkin, mixed, and continuous Galerkin methods for second order elliptic problems. *SIAM Journal on Numerical Analysis*, 47(2):1319–1365, 2009.

[15] Bernardo Cockburn, Jayadeep Gopalakrishnan, and Francisco-Javier Sayas. A projection-based error analysis of HDG methods. *Mathematics of Computation*, 79(271):1351–1367, 2010.

[16] Bernardo Cockburn, Johnny Guzmán, and Haiying Wang. Superconvergent discontinuous Galerkin methods for second-order elliptic problems. *Mathematics of Computation*, 78(265):1–24, 2009.

[17] Bernardo Cockburn, Fengyan Li, and Chi-Wang Shu. Locally divergence-free discontinuous Galerkin methods for the Maxwell equations. *Journal of Computational Physics*, 194(2):588–610, 2004.

[18] Gary Cohen, Xavier Ferrieres, and Sébastien Pernet. A spatial high-order hexahedral discontinuous Galerkin method to solve Maxwell's equations in time domain. *Journal of Computational Physics*, 217(2):340–363, 2006.

[19] Ronald Cools. Monomial cubature rules since "Stroud": a compilation—part 2. *Journal of Computational and Applied Mathematics*, 112(1-2):21–27, 1999.

[20] G. Cossarini, P. F. J. Lermusiaux, and C. Solidoro. Lagoon of Venice ecosystem: Seasonal dynamics and environmental guidance with uncertainty analyses and error subspace data assimilation. *Journal of Geophysical Research: Oceans*, 114(C6), June 2009.

[21] Eric Deleersnijder, Vincent Legat, and Pierre F. J. Lermusiaux. Multi-scale modelling of coastal, shelf and global ocean dynamics. *Ocean Dynamics*, 60(6):1357–1359, December 2010.

[22] Eric Deleersnijder and Pierre F. J. Lermusiaux. Multi-scale modeling: nested-grid and unstructured-mesh approaches. *Ocean Dynamics*, 58(5–6):335–336, December 2008.

[23] Paul F Dubois. Maintaining correctness in scientific programs. *Computing in Science & Engineering*, 7(3):80, 2005.

[24] Timothy F. Duda, Ying-Tsong Lin, W. Zhang, Bruce D. Cornuelle, and Pierre F. J. Lermusiaux. Computational studies of three-dimensional ocean sound fields in areas of complex seafloor topography and active ocean dynamics. In *Proceedings of the 10th International Conference on Theoretical and Computational Acoustics*, Taipei, Taiwan, 2011.

[25] Claes Eskilsson and Spencer J Sherwin. A triangular spectral/hp discontinuous Galerkin method for modelling 2d shallow water equations. *International Journal for Numerical Methods in Fluids*, 45(6):605–623, 2004.

[26] Holger Krekel et. al. `pytest`, 2019. Available at https://docs.pytest.org/en/latest/.

[27] Maurice S Fabien, Matthew G Knepley, Richard T Mills, and Beatrice M Riviere. Manycore Parallel Computing for a Hybridizable Discontinuous Galerkin Nested Multigrid Method. *SIAM Journal on Scientific Computing*, 41(2):C73–C96, 2019.

[28] C. Foucart, C. Mirabito, P. J. Haley, Jr., and P. F. J. Lermusiaux. Distributed implementation and verification of hybridizable discontinuous Galerkin methods for nonhydrostatic ocean processes. In *OCEANS Conference 2018*, Charleston, SC, October 2018. IEEE. In press.

[29] Francis X Giraldo, Jan S Hesthaven, and Tim Warburton. Nodal high-order discontinuous Galerkin methods for the spherical shallow water equations. *Journal of Computational Physics*, 181(2):499–525, 2002.

[30] Gene H Golub and Charles F Van Loan. *Matrix computations*, volume 3. JHU press, 2012.

[31] Siddharta Govindaraj. *Test-Driven Python Development*. Packt Publishing Ltd, 2015.

[32] Abhinav Gupta and Arun K Saha. Suppression of vortex shedding in flow around a square cylinder using control cylinder. *European Journal of Mechanics-B/Fluids*, 2019.

[33] P. J. Haley, Jr., A. Agarwal, and P. F. J. Lermusiaux. Optimizing velocities and transports for complex coastal regions and archipelagos. *Ocean Modeling*, 89:1–28, 2015.

[34] P. J. Haley, Jr., P. F. J. Lermusiaux, A. R. Robinson, W. G. Leslie, O. Logoutov, G. Cossarini, X. S. Liang, P. Moreno, S. R. Ramp, J. D. Doyle, J. Bellingham, F. Chavez, and S. Johnston. Forecasting and reanalysis in the Monterey Bay/California Current region for the Autonomous Ocean Sampling Network-II experiment. *Deep Sea Research Part II: Topical Studies in Oceanography*, 56(3–5):127–148, February 2009.

[35] Patrick J. Haley, Jr. and Pierre F. J. Lermusiaux. Multiscale two-way embedding schemes for free-surface primitive equations in the "Multidisciplinary Simulation, Estimation and Assimilation System". *Ocean Dynamics*, 60(6):1497–1537, December 2010.

[36] Jan S Hesthaven and Tim Warburton. *Nodal discontinuous Galerkin methods: algorithms, analysis, and applications*. Springer Science & Business Media, 2007.

[37] J. D. Hunter. Matplotlib: A 2D graphics environment. *Computing In Science & Engineering*, 9(3):90–95, 2007.

[38] Dominic Kempf and Timo Koch. System testing in scientific numerical software frameworks using the example of DUNE. *Archive of Numerical Software*, 5(1):151–168, 2017.

[39] Robert M Kirby. Visualization of discontinuous Galerkin based high-order methods. Technical report, University of Utah Salt Lake City, 2015.

[40] Robert M Kirby, Spencer J Sherwin, and Bernardo Cockburn. To CG or to HDG: a comparative study. *Journal of Scientific Computing*, 51(1):183–212, 2012.

[41] Martin Kronbichler, Katharina Kormann, and Wolfgang A Wall. Fast matrix-free evaluation of hybridizable discontinuous Galerkin operators. In *European Conference on Numerical Mathematics and Advanced Applications*, pages 581–589. Springer, 2017.

[42] Martin Kronbichler and Wolfgang A Wall. A performance comparison of continuous and discontinuous Galerkin methods with fast multigrid solvers. *SIAM Journal on Scientific Computing*, 40(5):A3423–A3448, 2018.

[43] Frans-Peter A. Lam, Patrick J. Haley, Jr., Jeroen Janmaat, Pierre F. J. Lermusiaux, Wayne G. Leslie, Mathijs W. Schouten, Lianke A. te Raa, and Michel Rixen. At-sea real-time coupled four-dimensional oceanographic and acoustic forecasts during Battlespace Preparation 2007. *Journal of Marine Systems*, 78(Supplement):S306–S320, November 2009.

[44] Andrea O Leone, Paola Marzano, Enrico Gobbetti, Riccardo Scateni, and Sergio Pedinotti. Discontinuous finite element visualization. In *Proceedings 8th International Symposium on Flow Visualization*, 1998.

[45] P. F. J. Lermusiaux, C.-S. Chiu, G. G. Gawarkiewicz, P. Abbot, A. R. Robinson, R. N. Miller, P. J. Haley, Jr, W. G. Leslie, S. J. Majumdar, A. Pang, and F. Lekien. Quantifying uncertainties in ocean predictions. *Oceanography*, 19(1):92–105, 2006.

[46] P. F. J. Lermusiaux, P. J. Haley, W. G. Leslie, A. Agarwal, O. Logutov, and L. J. Burton. Multiscale physical and biological dynamics in the Philippine Archipelago: Predictions and processes. *Oceanography*, 24(1):70–89, 2011. Special Issue on the Philippine Straits Dynamics Experiment.

[47] P. F. J Lermusiaux, P. J. Haley, Jr, and N. K. Yilmaz. Environmental prediction, path planning and adaptive sampling: sensing and modeling for efficient ocean monitoring, management and pollution control. *Sea Technology*, 48(9):35–38, 2007.

[48] P. F. J. Lermusiaux, D. N. Subramani, J. Lin, C. S. Kulkarni, A. Gupta, A. Dutt, T. Lolla, P. J. Haley, Jr., W. H. Ali, C. Mirabito, and S. Jana. A future for intelligent autonomous ocean observing systems. *Journal of Marine Research*, 75(6):765–813, November 2017. The Sea. Volume 17, The Science of Ocean Prediction, Part 2.

[49] Pierre F. J. Lermusiaux. Numerical fluid mechanics. MIT OpenCourseWare, May 2015.

[50] W. G. Leslie, A. R. Robinson, P. J. Haley, Jr, O. Logutov, P. A. Moreno, P. F. J. Lermusiaux, and E. Coelho. Verification and training of real-time forecasting of multi-scale ocean dynamics for maritime rapid environmental assessment. *Journal of Marine Systems*, 69(1):3–16, 2008.

[51] Wai-Hung Liu and Andrew H Sherman. Comparative analysis of the Cuthill–McKee and the reverse Cuthill–McKee ordering algorithms for sparse matrices. *SIAM Journal on Numerical Analysis*, 13(2):198–213, 1976.

[52] Anders Logg, Kent-Andre Mardal, and Garth Wells. *Automated solution of differential equations by the finite element method: The FEniCS book*, volume 84. Springer Science & Business Media, 2012.

[53] A. Meurer, C.P. Smith, M. Paprocki, et al. `SymPy`: symbolic computing in Python. *PeerJ Computer Science*, 3:e103, Jan 2017.

[54] Miriah Meyer, Blake Nelson, Robert Kirby, and Ross Whitaker. Particle systems for efficient and accurate high-order finite element visualization. *IEEE Transactions on Visualization and Computer Graphics*, 13(5):1015–1026, 2007.

[55] MSEAS Group. MSEAS Software, 2013.

[56] Ngoc Cuong Nguyen and Jaume Peraire. Hybridizable discontinuous Galerkin methods for partial differential equations in continuum mechanics. *Journal of Computational Physics*, 231(18):5955–5988, 2012.

[57] Ngoc Cuong Nguyen and Jaume Peraire. Lecture notes for MIT course 16.930, Spring 2017.

[58] Ngoc Cuong Nguyen, Jaume Peraire, and Bernardo Cockburn. An implicit high-order hybridizable discontinuous Galerkin method for linear convection–diffusion equations. *Journal of Computational Physics*, 228(9):3232–3254, 2009.

[59] Ngoc Cuong Nguyen, Jaume Peraire, and Bernardo Cockburn. An implicit high-order hybridizable discontinuous Galerkin method for nonlinear convection–diffusion equations. *Journal of Computational Physics*, 228(23):8841–8855, 2009.

[60] Ngoc Cuong Nguyen, Jaume Peraire, and Bernardo Cockburn. High-order implicit hybridizable discontinuous Galerkin methods for acoustics and elastodynamics. *Journal of Computational Physics*, 230(10):3695–3718, 2011.

[61] Ngoc Cuong Nguyen, Jaume Peraire, and Bernardo Cockburn. An implicit high-order hybridizable discontinuous Galerkin method for the incompressible Navier–Stokes equations. *Journal of Computational Physics*, 230(4):1147–1170, 2011.

[62] Brian Okken. *Python Testing with Pytest: Simple, Rapid, Effective, and Scalable*. Pragmatic Bookshelf, 2017.

[63] Reiner Onken, Allan R. Robinson, Pierre F. J. Lermusiaux, Patrick J. Haley, and Larry A. Anderson. Data-driven simulations of synoptic circulation and transports in the Tunisia-Sardinia-Sicily region. *Journal of Geophysical Research: Oceans*, 108(C9), 2003.

[64] Jaime Peraire, Ngoc Nguyen, and Bernardo Cockburn. A hybridizable discontinuous Galerkin method for the compressible Euler and Navier-Stokes equations. In *48th AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition*, page 363, 2010.

[65] Harry Percival. *Test-driven development with Python: obey the testing goat; using Django, Selenium, and JavaScript*. O'Reilly Media, Inc., 2014.

[66] Steven R. Ramp, Pierre F. J. Lermusiaux, Igor Shulman, Yi Chao, Rebecca E. Wolf, and Frederick L. Bahr. Oceanographic and atmospheric conditions on the continental shelf north of the Monterey Bay during August 2006. *Dynamics of Atmospheres and Oceans*, 52(1–2):192–223, September 2011. Special issue of Dynamics of Atmospheres and Oceans in honor of Prof. A. R. Robinson.

[67] William H Reed and TR Hill. Triangular mesh methods for the neutron transport equation. Technical report, Los Alamos Scientific Lab., N. Mex.(USA), 1973.

[68] Jean-François Remacle, Nicolas Chevaugeon, Emilie Marchandise, and Christophe Geuzaine. Efficient visualization of high-order finite elements. *International Journal for Numerical Methods in Engineering*, 69(4):750–771, 2007.

[69] Yves Reymen, Martine Baelmans, and Wim Desmet. Study of convergence and efficiency of a nodal quadrature-free discontinuous Galerkin method on meshes of tetrahedral and hexahedral elements. In *International Conference On Spectral and High Order Methods, Beijing, China*, 2007.

[70] Beatrice Riviere. *Discontinuous Galerkin methods for solving elliptic and parabolic equations: theory and implementation*. SIAM, 2008.

[71] Patrick J Roache. Code verification by the method of manufactured solutions. *Journal of Fluids Engineering*, 124(1):4–10, 2002.

[72] Patrick J Roache and Stanly Steinberg. Symbolic manipulation and computational fluid dynamics. *AIAA journal*, 22(10):1390–1394, 1984.

[73] Xevi Roca, Cuong Nguyen, and Jaime Peraire. Scalable parallelization of the hybridized discontinuous Galerkin method for compressible flow. In *21st AIAA Computational Fluid Dynamics Conference*, page 2939, 2013.

[74] Xevi Roca, Ngoc Cuong Nguyen, and Jaime Peraire. GPU-accelerated sparse matrix-vector product for a hybridizable discontinuous Galerkin method. In *49th AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition*, page 687, 2011.

[75] Yousef Saad. *Iterative methods for sparse linear systems*, volume 82. SIAM, 2003.

[76] Filip Sadlo, Markus Üffinger, Christian Pagot, Daniel Osmari, João Comba, Thomas Ertl, Claus-Dieter Munz, and Daniel Weiskopf. Visualization of cell-based higher-order fields. *Computing in Science & Engineering*, 13(3):84–91, 2011.

[77] Will J Schroeder, Bill Lorensen, and Ken Martin. *The visualization toolkit: an object-oriented approach to 3D graphics*. Kitware, 2004.

[78] Jonathan Richard Shewchuk et al. An introduction to the conjugate gradient method without the agonizing pain, 1994.

[79] Ioannis Toulopoulos and John A Ekaterinaris. High-order discontinuous Galerkin discretizations for computational aeroacoustics in complex domains. *AIAA journal*, 44(3):502–511, 2006.

[80] John A Trangenstein. Scientific visualization. In *Scientific Computing*, pages 291–327. Springer, 2017.

[81] Lloyd N Trefethen and David Bau III. *Numerical linear algebra*, volume 50. SIAM, 1997.

[82] M. P. Ueckermann. Towards next generation ocean models: Novel discontinuous Galerkin schemes for 2D unsteady biogeochemical models. Master's thesis, Massachusetts Institute of Technology, Department of Mechanical Engineering, Cambridge, MA, September 2009.

[83] M. P. Ueckermann. *High Order Hybrid Discontinuous Galerkin Regional Ocean Modeling*. PhD thesis, Massachusetts Institute of Technology, Department of Mechanical Engineering, Cambridge, MA, February 2014.

[84] M. P. Ueckermann and P. F. J. Lermusiaux. High order schemes for 2D unsteady biogeochemical ocean models. *Ocean Dynamics*, 60(6):1415–1445, December 2010.

[85] M. P. Ueckermann and P. F. J. Lermusiaux. Hybridizable discontinuous Galerkin projection methods for Navier–Stokes and Boussinesq equations. *Journal of Computational Physics*, 306:390–421, 2016.

[86] M. P. Ueckermann, C. Mirabito, P. J. Haley, Jr., and P. F. J. Lermusiaux. High order hybridizable discontinuous Galerkin projection schemes for non-hydrostatic physical-biogeochemical ocean modeling. *Ocean Dynamics*, 2019. To be submitted.

[87] Greg Wilson, Dhavide A Aruliah, C Titus Brown, Neil P Chue Hong, Matt Davis, Richard T Guy, Steven HD Haddock, Kathryn D Huff, Ian M Mitchell, Mark D Plumbley, et al. Best practices for scientific computing. *PLoS biology*, 12(1):e1001745, 2014.

[88] Yulong Xing, Xiangxiong Zhang, and Chi-Wang Shu. Positivity-preserving high order well-balanced discontinuous Galerkin methods for the shallow water equations. *Advances in Water Resources*, 33(12):1476–1493, 2010.

[89] J. Xu, P. F. J. Lermusiaux, P. J. Haley Jr., W. G. Leslie, and O. G. Logutov. Spatial and Temporal Variations in Acoustic propagation during the PLUSNet-07 Exercise in Dabob Bay. In *Proceedings of Meetings on Acoustics (POMA)*, volume 4, page 11. Acoustical Society of America 155th Meeting, 2008.

[90] Mengping Zhang and Chi-Wang Shu. An analysis of three different formulations of the discontinuous Galerkin method for diffusion equations. *Mathematical Models and Methods in Applied Sciences*, 13(03):395–413, 2003.

[91] OC Zienkiewicz and CJ Parekh. Transient field problems: two-dimensional and three-dimensional analysis by isoparametric finite elements. *International Journal for Numerical Methods in Engineering*, 2(1):61–71, 1970.